НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення

на тему Інформаційно-довідкова система супроводження процесу експлуатації лабораторних стендів

Виконав : студент 4 курсу, групи ТІ-61
Кондрашов Ігор Олександрович _____
(підпис)

Керівник :
Викладач, к.ф.м.н, доцент Тарнавський Юрій Адамович _____
(підпис)

Рецензент :
Викладач, к.т.н, доцент Новаківський Євген Валерійович _____
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

__Кондрашову Ігорю Олександровичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи ____Інформаційно-довідкова система супроводження процесу експлуатації лабораторних стендів
керівник роботи к.ф.м.н, доцент Тарнавський Юрій Адамович
затверджена наказом вищого навчального закладу від ”25” травня 2020р.
№ 1168-с

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи Javascript, PHP, Vue.js, Laravel,
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Порівняльний аналіз існуючих рішень, вибір методів вирішення задачі, проектування системи, розробка програмного продукту.
5. Перелік ілюстративного матеріалу
_____
_____
_____
_____
_____

7. Дата видачі завдання ”___”_____201___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	18.02.2020	
2.	Вивчення та аналіз задачі	19.02.2020 – 05.03.2020	
3.	Розробка архітектури та загальної структури системи	06.03.2020 – 31.03.2020	
4.	Розробка структур окремих підсистем	01.04.2020 – 15.04.2020	
5.	Програмна реалізація системи	16.04.2020 – 15.05.2020	
6.	Оформлення пояснювальної записки	16.04.2020 – 15.05.2020	
7.	Захист програмного продукту	16.05.2020	
8.	Передзахист	11.06.2020	
9.	Захист	17.06.2020	

Студент _____ Кондрашов І.О.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Тарнавський Ю.А.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

«Інформаційно-довідкова система супроводження процесу експлуатації лабораторних стендів». КПІ ім. Ігоря Сікорського, Київ, 2020.

Дипломний проект присвячений розробці системи, що слугуватиме для моніторингу експлуатації лабораторних стендів, ведення обліку деталей лабораторій, стендів та виведення статистичних даних на екран.

Текстова документація містить інформацію про розроблені алгоритми, застосовані шаблони проектування, розробку та тестування. Також в документації детально описаний обраний для реалізації проекту стек технологій.

Ключові слова: стенди, лабораторії, деталі, експлуатація, Laravel, Vue.js, Javascript, REST API.

Розмір пояснювальної записки – 61 аркуш, містить 32 ілюстрацій.

ABSTRACT

"Information system for monitoring the exploitation of laboratory stands." NTUU KPI Igor Sikorsky, Kyiv, 2020.

The diploma project is devoted to the development of a system that will serve to monitor the operation of laboratory stands, record accounting of laboratory parts, stands and display statistics.

The text documentation contains information about the developed algorithms, applied design template, development and testing. The documentation also describes in detail the technology stack selected for project implementation.

Keywords: stands, laboratories, details, operation, Laravel, Vue.js, Javascript, REST API.

The size of the explanatory note - 61 pages, contains 32 illustrations.

Зміст

1. Перелік термінів та скорочень

1. Middleware – функція проміжної обробки, забезпечує зручний механізм фільтрації HTTP-запитів, що надходять у програму.
2. REST API – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів.
3. Паттерн (шаблон) проектування – ефективний спосіб вирішення типової задачі проектування програмного забезпечення.
4. Лабораторний стенд - повірене лабораторне або виробниче обладнання, яке призначене для спеціальних, контрольних, приймальних випробувань.
5. JSON (JavaScript Object Notation) — текстовий формат обміну структурованих даних між комп'ютерами.
6. MVVM (Model-View-ViewModel) – шаблон проектування модель-вигляд-модель вигляду.
7. ORM (Object-relational mapping) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».
8. UML (Unified Modeling Language) – уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування.
9. CRUD (Create Read Update Delete) – чотири основні функції керування даними: створення, зчитування, зміна і видалення.
10. API (Application Programming Interface) – прикладниий програмний інтерфейс (інтерфейс програмування застосунків, інтерфейс прикладного програмування) набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.
11. Canvas - це HTML елемент, що використовується для малювання графіки засобами мов програмування (зазвичай це JavaScript). Він може, наприклад,

використовуватися для малювання графів, створення колажів або простої (і не дуже) анімації.

12. MVC (Model View Controller) – шаблон проектування модель-вигляд-контроллер.

2. Вступ

В сучасних лабораторіях при університетах часто відсутній моніторинг експлуатації стендів, та їх складових. Тому часто стенди неочікувано виходять з ладу, необхідних запчастин не вистачає, немає якісного обліку цих запчастин і вони самі розкидані по лабораторії.

Як показав попередній аналіз – у відкритому доступі не існує програмного забезпечення, яке покриває вирішення подібних проблем, тому буде актуальною розробка інформаційно-довідкової системи супроводження процесу експлуатації лабораторних стендів. Існують інформаційно-довідкова системи, які за своїм загальним сенсом є подібними до розроблюваної системи, проте мають іншу предметну область, або ж навпаки є занадто узагальненими. Саме тому є необхідність в розробці системи яка буде задовольняти потреби лабораторій.

В наступних розділах буде детальніше описано весь функціонал системи, засоби розробки та вимоги до використання. Також буде подано детальний опис програмної реалізації, структура і архітектура програмного забезпечення і використаний стек технологій.

3. Задачі розробки інформаційно-довідкової системи

Метою роботи є розробка інформаційно-довідкової система супроводження процесу експлуатації лабораторних стендів. Ця система має бути масштабованою для використання на різних платформах, саме тому розроблений застосунок буде веб-системою. Система матиме просту реєстрацію і авторизацію, після якої користувачі матимуть змогу переглядати повний список стендів які розроблені в лабораторії. Можна буде переглядати інформацію про запчастини стенду, їхній статус (партію, термін придатності тощо). Також користувач матиме змогу самостійно створити стенд, додати до нього всі використані запчастини, і в подальшому редагувати його стан. При виведенні стенда з експлуатації його можна буде видалити з веб системи.

Розроблена система матиме практичне застосування, зокрема її можна буде використовувати при лабораторіях університетів, або навіть невеликих підприємств. Інформаційно-довідкова система повинна мати потенціал до розширення, і до введення нового функціоналу.

Потенційними користувачами цієї системи стануть лабораторії при університетах, а також дрібні підприємства, які використовують просте технічне обладнання.

Для входу в розроблену систему необхідно створити свій аккаунт, який буде в подальшому використовуватись особисто, або для групи осіб чи лабораторії. Для цього просто необхідно вказати свій логін і пароль на початковій сторінці реєстрації.

Далі можна розділити розроблену систему на три підзадачі. Перша — це ведення моніторингу та статистики всіх присутніх деталей лабораторії чи підприємства. У ній користувач може добавляти нові деталі, редагувати їхню інформацію, видаляти їх, а також відслідковувати загальну статистику: кількість деталей одного виду, їхню справність і т.д.. Друга підзадача — це моніторинг самих стендів. В цьому розділі ми можемо переглянути з яких деталей

складаються наші стенди, а також загальний стан експлуатації цих стендів. Для цього розглядається варіант підключення сторонніх API. Третя підзадача — це розташування стендів та запчастин в межах лабораторії. Для цього користувач може створити загальну схему своєї лабораторії та розставити наявні елементи в її межах.

Для реалізації такої системи найбільш зручним рішенням є SPA застосунок, який дасть швидку навігацію в межах системи, та асинхронне підвантаження даних.

4. Аналіз систем моніторингу експлуатації

Пошук систем моніторингу експлуатації технічних об’єктів показав, що аналогічних систем у відкритому доступі немає. Проте є системи, які займаються моніторингом технічних об’єктів. Вони надають наступні послуги:

1. Віддалений моніторинг системи — надання набору інструментів для обслуговування системи власною службою експлуатації.
2. Безперервний контроль, за станом системи енергозабезпечення.
3. Віддалене управління процесами запуску і синхронізації станцій.
4. Надання доступу до журналу подій з будь-якого місця, де б не знаходився фахівець.

Встановлення системи моніторингу на об’єкті відкриває багато можливостей, зокрема дозволяє:

1. Забезпечити збір інформації в реальному часі про роботу установки.
2. Своєчасно відстежити зміни в роботі і скорегувати програму експлуатації та обслуговування обладнання.
3. Передбачити і вирішити технічні проблеми, що виникають.
4. Завчасно спланувати постачання необхідних запчастин і витратних матеріалів.
5. Знизити витрати на подальше обслуговування установки.

Такі послуги пропонує компанія “ Dalgakiran”[1], яка спеціалізується на продажі, оренжі, та обслуговуванням технічного обладнання на виробництвах. Проте програмне забезпечення пропоноване компанією має ряд недоліків, які роблять його таким, що не задовольняє умови поставленої задачі.

По-перше, цільовою аудиторією компанії “ Dalgakiran” є підприємства, які використовують технічне обладнання самої компанії. Тобто дане програмне забезпечення не розповсюджується на технічне обладнання інших компаній.

По-друге, для встановлення такого програмного забезпечення необхідне втручання сторонніх спеціалістів. Ви не зможете самостійно інтегрувати таку систему без допомоги працівників компанії.

По-третє, система моніторингу експлуатації лабораторних стендів розробляється в першу чергу для некомерційного використання (для використання лабораторіями університетів). Програмного забезпечення для моніторингу експлуатації об’єктів від компанії “ Dalgakiran” немає у відкритому доступі. Це програмне забезпечення передбачає комерційне використання.

Зважаючи всі вище перелічені пункти, дане програмне забезпечення не задовольняє вимог поставленого завдання.

5. Засоби розробки

Інформаційно-довідкова система моніторингу експлуатації лабораторних стендів повинна виконувати умову кросплатформенності, тому було прийнято рішення зробити її веб-застосунком. Також наша система буде SPA застосунком, для того щоб полегшити взаємодію користувача з системою, та спростити написання клієнтської частини. Через це був обраний javascript фреймворк Vue.js, який відповідає поставленим вимогам. Також для клієнтської частини був використаний плагін Vuetify, який значно полегшує роботу з даними на стороні клієнта, а також задає дизайн елементів.

Що стосується серверної частини, то вона представлена у вигляді REST API, оскільки це найзручніший і найпоширеніший вид зв'язку між сервером і клієнтською частиною у випадку використання SPA. Також використання REST API дасть нам більше простору до розширення системи в сторону мобільного додатку, оскільки клієнтська і серверна частина у нас живуть окремим життям. Вибір технологій для написання REST API є дуже широким, проте був обраний PHP фреймворк Laravel, оскільки він з коробки дає нам можливість працювати з Vue.js, а також надає свої готові функції що стосуються авторизації користувача в системі через токени, і має зручну ORM для роботи з базою даних.

Стосовно останньої, то поставленим задачам цілком відповідає база даних MySQL, а також як вже було сказано, Laravel надає нам зручну ORM для цієї системи керування базою даних.

5.1. JavaScript

JavaScript (JS) [11] — це невибаглива до ресурсів мова програмування з функціями першого класу, код якої інтерпретується та компілюється під час виконання. Хоча JavaScript насамперед відома як скриптова мова для веб-сторінок, вона також використовується у багатьох небраузерних середовищах

на кшталт Node.js, Apache CouchDB та Adobe Acrobat. JavaScript — прототип-орієнтована динамічна мова, що має декілька парадигм та підтримує об'єктно-орієнтований, імперативний та декларативний (тобто функціональне програмування) стилі.

Стандартом для JavaScript є ECMAScript. Станом на 2012 рік усі сучасні браузери вже мали повну підтримку ECMAScript 5.1. Застарілі браузери підтримують щонайменше ECMAScript 3. 17 червня 2015 року ECMA International випустила шосту базову версію ECMAScript з офіційною назвою ECMAScript 2015, яка у попередніх обговореннях іменувалася ECMAScript 6 або ES6. Відтоді стандарти ECMAScript оновлюються раз на рік.

Сьогодні JavaScript може виконуватися не тільки в браузері, а й на сервері або на будь-якому іншому пристрої, який має спеціальну програму, що називається рушієм JavaScript [16].

У браузера є власний рушій, який іноді називають «віртуальна машина JavaScript».

Різні рушії мають різні «кодові імена». наприклад:

- V8 - в Chrome і Opera.
- SpiderMonkey - в Firefox.
- «Trident» і «Chakra» для різних версій IE, «ChakraCore» для Microsoft Edge, «Nitro» і «SquirrelFish» для Safari і т.д.

Це важливий пункт, оскільки для розробки на Vue.js код буде попередньо компілюватися на платформі Node.js, що побудована на рушієві V8.

5.2. PHP

PHP (Hypertext Preprocessor) [12], попередня назва: Personal Home Page Tools — скриптова мова програмування, була створена для генерації HTML-сторінок на стороні веб-сервера. PHP є однією з найпоширеніших мов, що використовуються у сфері веб-розробок (разом із Java, .NET, Perl, Python, Ruby).

PHP підтримується переважною більшістю хостинг-провайдерів. PHP — проект відкритого програмного забезпечення.

PHP інтерпретується веб-сервером у HTML-код, який передається на сторону клієнта. На відміну від скриптової мови JavaScript, користувач не бачить PHP-коду, тому що браузер отримує готовий html-код. Це є перевагою з точки зору безпеки, але погіршує інтерактивність сторінок. Але ніхто не забороняє використовувати PHP для генерування JavaScript-кодів, які виконуються вже на стороні клієнта.

Популярність в області створення веб-сайтів визначає наявність великого набору вбудованих середовищ і додаткових модулів для розробки веб-застосунків. Основні з них:

- робота з cookies і сесіями;
- автоматичне вилучення POST- і GET-параметрів, а також змінних оточення веб-сервера в зумовлені масиви;
- автоматизоване надсилання HTTP-заголовків;

5.3. Vue.js

Vue.js[2] — JavaScript-фреймворк для створення інтерфейсів користувача на основі моделей даних, через реактивне зв'язування даних. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня відображення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками.

Одна із найвиразніших особливостей Vue — це ненав'язлива реактивна система. Моделі це просто плоскі JavaScript об'єкти. Це робить керування станами

дуже простим та інтуїтивним. Vue надає оптимізований ре-рендеринг з коробки без потреби робити що-небудь додатково. Кожен компонент слідкує за своїми реактивними залежностями під час рендерингу, тому система знає точно коли має відбуватись ре-рендеринг і які компоненти потрібно ре-рендерити.

5.4. Laravel

Laravel[4] — безкоштовний, з відкритим кодом PHP-фреймворк, створений і призначений для розробки веб-додатків відповідно до шаблону model–view–controller (MVC). Деякими з особливостей Laravel є модульна система з виділеним менеджером залежностей Composer, різні способи для доступу до реляційних баз даних, утиліти, які допомагають в розгортанні додатків і технічного обслуговування, а також його орієнтація на синтаксичний цукор.

Як і будь-який PHP фреймворк, Laravel має безліч функцій, які виділяються серед інших. Ось деякі, які найбільш важливі.

1. Пакети (bundles) в Laravel це як PEAR для PHP. Вони є доповненнями, які ви можете завантажити та підключити до Laravel. На даний момент існує досить багато загальнодоступних пакетів в репозиторії пакетів Laravel, який постійно поповнюється. Laravel дозволяє використовувати командний рядок для швидкого встановлення пакетів.
2. ORM — це технологія програмування, що розроблена для полегшення програмістам роботи з БД шляхом надання методів API для типових операцій (вибірка, додавання, оновлення, видалення і т.д.). Eloquent ORM є найбільш передовою ActiveRecord реалізацією для PHP. Подібно Doctorine ORM дозволяє зробити будь-яку роботу з базою даний дуже простий. Можна з легкістю виконувати CRUD операції над вашою базою даних. А так же створювати різні зв'язки між таблицями.
3. Міграції бази даних є вельми корисні для будь-якого проекту, особливо для проектів з декількома розробниками, дозволяючи мати останню версію бази

даних у всіх розробників. У Laravel для цього потрібна лише одна команда в командному рядку. У Laravel є власний конструктор таблиць, що дозволяє швидко писати зміни в базу даних.

5.5. MySQL

MySQL — вільна система керування реляційними базами даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

MySQL — компактний багатопотоковий сервер баз даних. Характеризується високою швидкістю, стійкістю і простотою використання.

MySQL вважається гарним рішенням для малих і середніх застосувань. Сирцеві коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатопоточності, що підвищує продуктивність системи в цілому.

Можливості сервера MySQL:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн;
- висока швидкість виконання команд;
- наявність простої і ефективної системи безпеки.

5.6. Додаткові плагіни та бібліотеки

Також для зручності розробки були використані деякі додаткові плагіни і бібліотеки. Плагін — це незалежний програмний модуль, що динамічно підключається до основної програми, призначений для розширення або використання її можливостей. Тобто це невелике розширення, яке, як правило, не задає певної структури для розробки, на відміну від фреймворків. Проте варто їх згадати, так як вони є невід'ємною частиною розробленої веб-системи.

Vuetify[5] — бібліотека Vue UI з красивими рукотворними матеріальними компонентами. Не потрібні навички дизайну - все, що потрібно для створення дивовижних додатків, є у вас під рукою.

Chart.js — плагін для побудови графіків. Є дуже гнучким інструментом, який робить побудову дуже простою та дозволяє анімувати зміни.

Interract.js — плагін для реалізації drag and drop функціоналу. Використовується для симуляції приміщення лабораторії. З цим плагіном дуже просто реалізувати функціонал перетягування DOM елементів в межах одного контейнера.

Laravel спрощує аутентифікацію за допомогою традиційних форм входу, а як щодо API? API зазвичай використовують токени для автентифікації користувачів і не підтримують стан сесії між запитами. Laravel робить автентифікацію API легкою в реалізації за допомогою Laravel Passport. Цей пакет є офіційним пакетом, який підтримується фреймворком Laravel.

5.7. Canvas

Для створення графіків на сторінці одного стенда була використана технологія canvas.

Елемент HTML canvas використовується для малювання графіки під час руху через JavaScript. Елемент canvas - це лише контейнер для графіки. Для

малювання графіки потрібно використовувати JavaScript. Полотно canvas має кілька методів для малювання контурів, коробок, кіл, тексту та додавання зображень. Елемент canvas підтримується усіма поширеними браузерами (рисунок 5.1.)

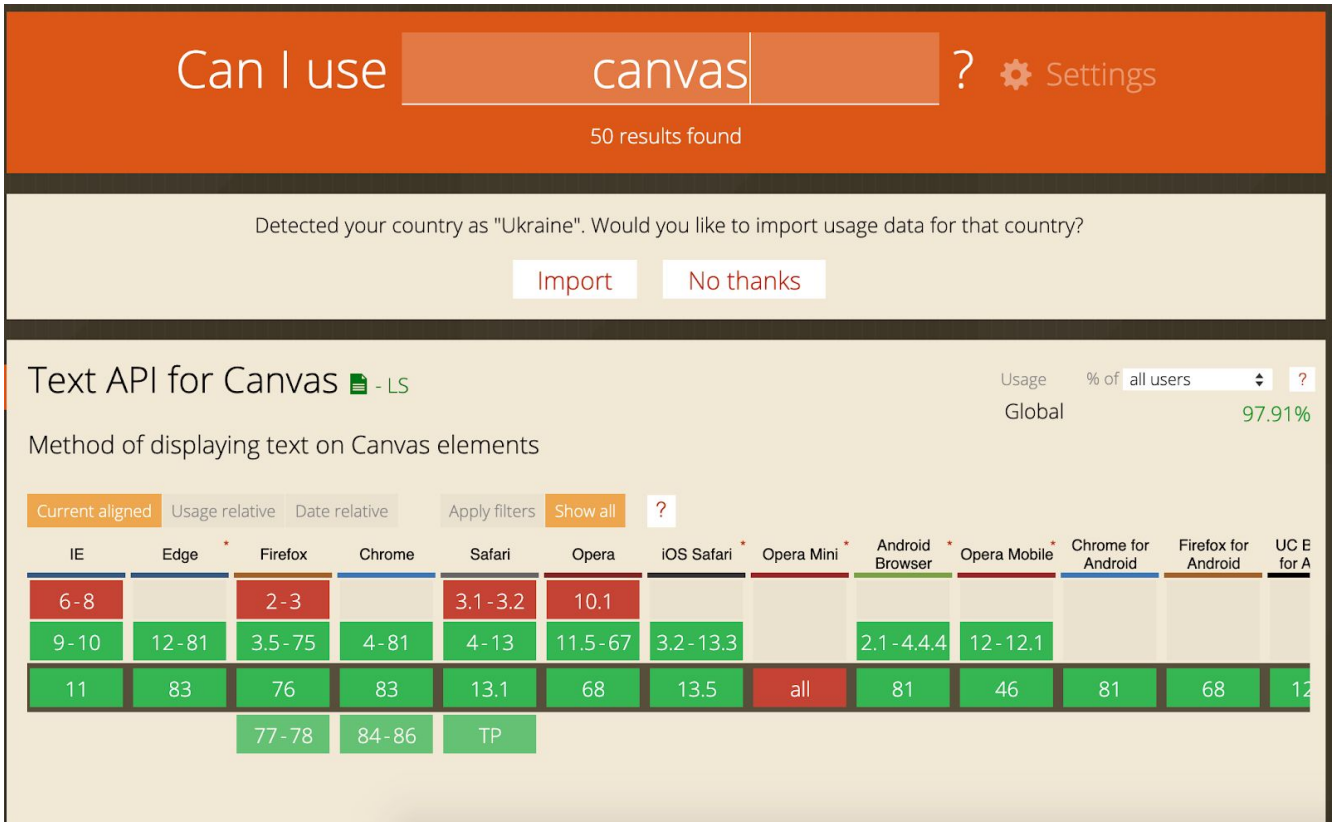

Рисунок 5.1. – підтримка елемент canvas браузерами

Єдиний браузер який не підтримує технологію canvas – Opera mini. Проте загальна кількість користувачів цього браузера становить менше 1% серед усіх сучасних браузерів.

5.8. Системні вимоги

Оскільки дана система є веб-системою, то немає особливих вимог до програмного забезпечення пристрою користувача. Система може використовуватись на різних операційних системах. Сама система не потребує інсталяції, єдиною вимогою є наявність браузера. Vue не підтримує браузер Internet Explorer 8 і нижче, так як використовує можливості ECMAScript 5, які

неможливо емулювати в IE8. В іншому, підтримуються всі браузери, сумісні з ECMAScript 5, наприклад Google Chrome, Mozilla Firefox, Safari, Opera і тд.

Для роботи з Vue.js фреймворком використовується платформа Node.js, а для розгортання проекту на Laravel – веб-сервер Apache. Відповідні платформи необхідні для розгортання проекту на віддаленому сервері.

Також підтримку використаних у проекті технологій браузерами можна перевірити на веб-ресурсі CanIUse [15]. Наприклад можна перевірити технологію localStorage, що використовується в проекті для збереження токена авторизації. Результат перевірки зображено на рисунку 5.2.

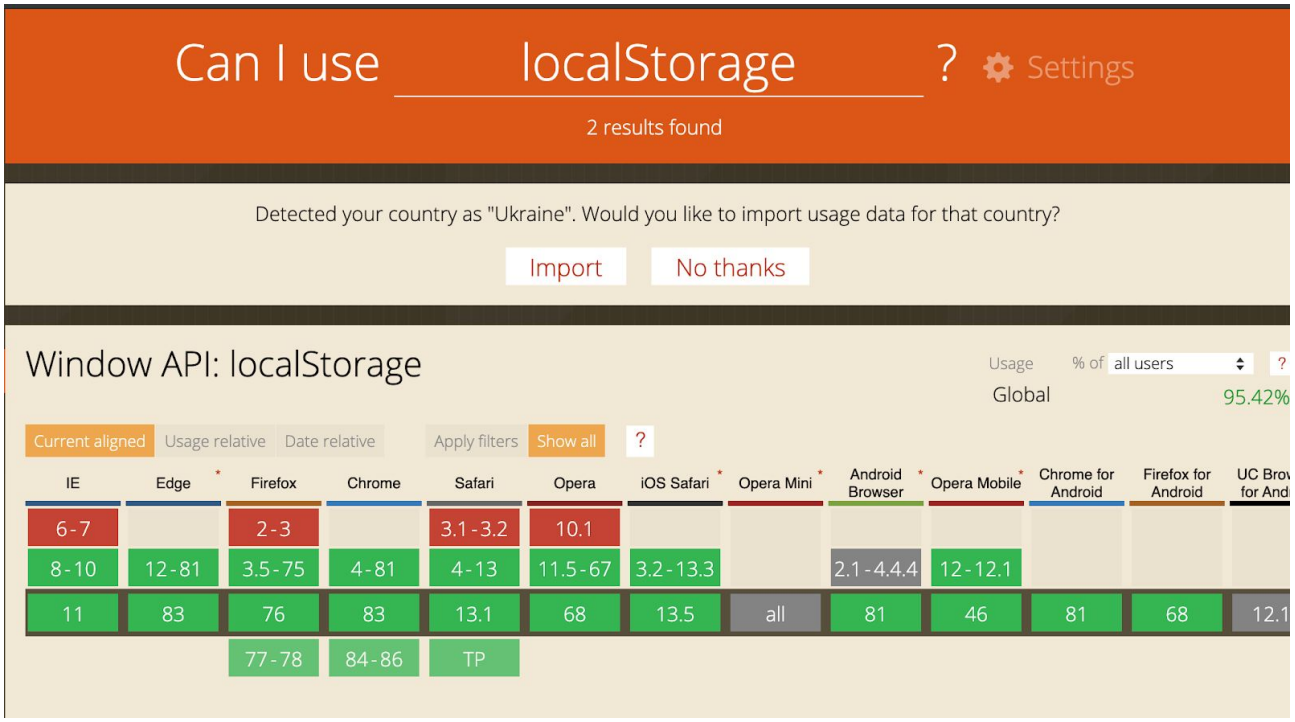

Рисунок 5.2. – перевірка localStorage на підтримку браузерів.

З рисунка видно, що більшість браузерів підтримують дану технологію, і лише від невеликою кількості браузерів інформація відсутня (браузери позначені сірим кольором).

6. Проектування системи

Перед тим як перейти до написання програмного коду продукту необхідно правильно визначити його структуру, для цього власне і використовуються UML-діаграми.

Уніфікована мова моделювання (UML) [9] - це стандартна мова візуального моделювання, призначена для:

- моделювання бізнесу та подібних процесів;
- аналізу, проектування та впровадження програмних систем.

UML - це загальна мова для бізнес-аналітиків, архітекторів програмного забезпечення та розробників, яка використовується для опису, конкретизації, проектування та документування існуючих чи нових бізнес-процесів, структури та поведінки артефактів програмних систем.

Першою розробленою для системи діаграмою є діаграма прецедентів. Діаграма прецедентів — в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання. Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із

системою за допомогою так званих варіантів використання. Варіант використання використовують для опису послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

Розглянемо діаграму прецедентів для нашої системи (рисунок 6.1).

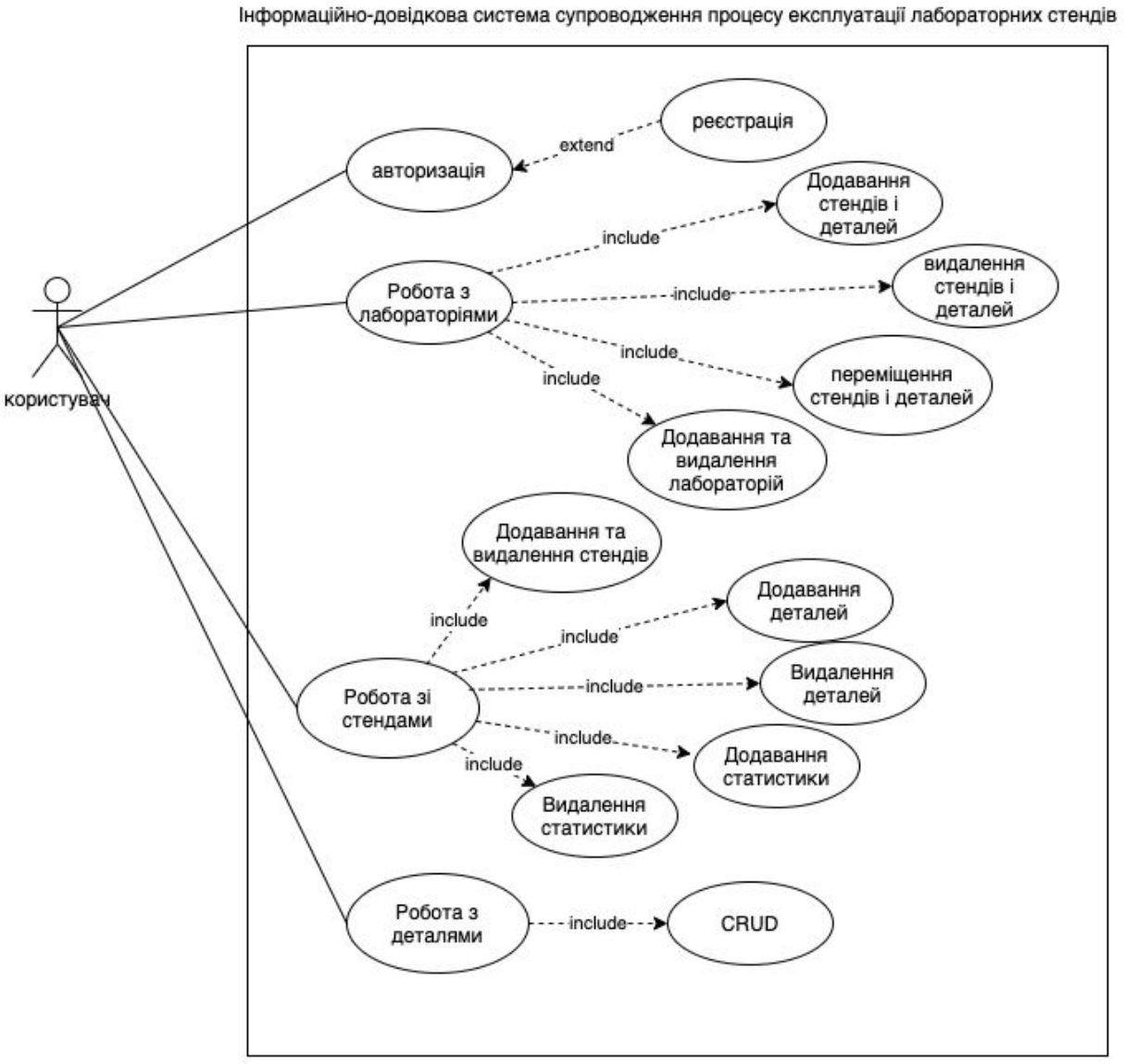

Рисунок 6.1 – діаграма прецедентів

Єдиним актором нашої системи є сам її користувач.

Першим прецедентом діаграми є авторизація користувача, і має відношення розширення з реєстрацією, оскільки кожен користувач мусить зареєструватися хоча б один раз, і після того вже може здійснювати багаторазовий вхід в систему.

Далі ми бачимо три прецеденти, які описують всю суть роботи в системі. Кожен користувач може працювати з лабораторіями, стендами та деталями. Робота з лабораторіями та стендами має зв’язок включення з двома операціями – створення та видалення. Це базовий функціонал який можна застосувати до цих двох сутностей. Також робота з лабораторіями має зв’язок включення з додаванням і видаленням стендів та деталей, а також їх переміщення в межах лабораторії. Робота зі стендами має зв’язок включення з прецедентами додавання та видаленням деталей, а також додаванням та видаленням статистики.

Останнім прецедентом діаграми є робота з деталями. Цей прецедент має зв’язок включення з CRUD операціями.

Наступна діаграма – діаграма поекранного відображення (рисунок 6.2). Вона не підпорядковується специфікації UML, проте найкраще демонструє порядок роботи користувача з системою.

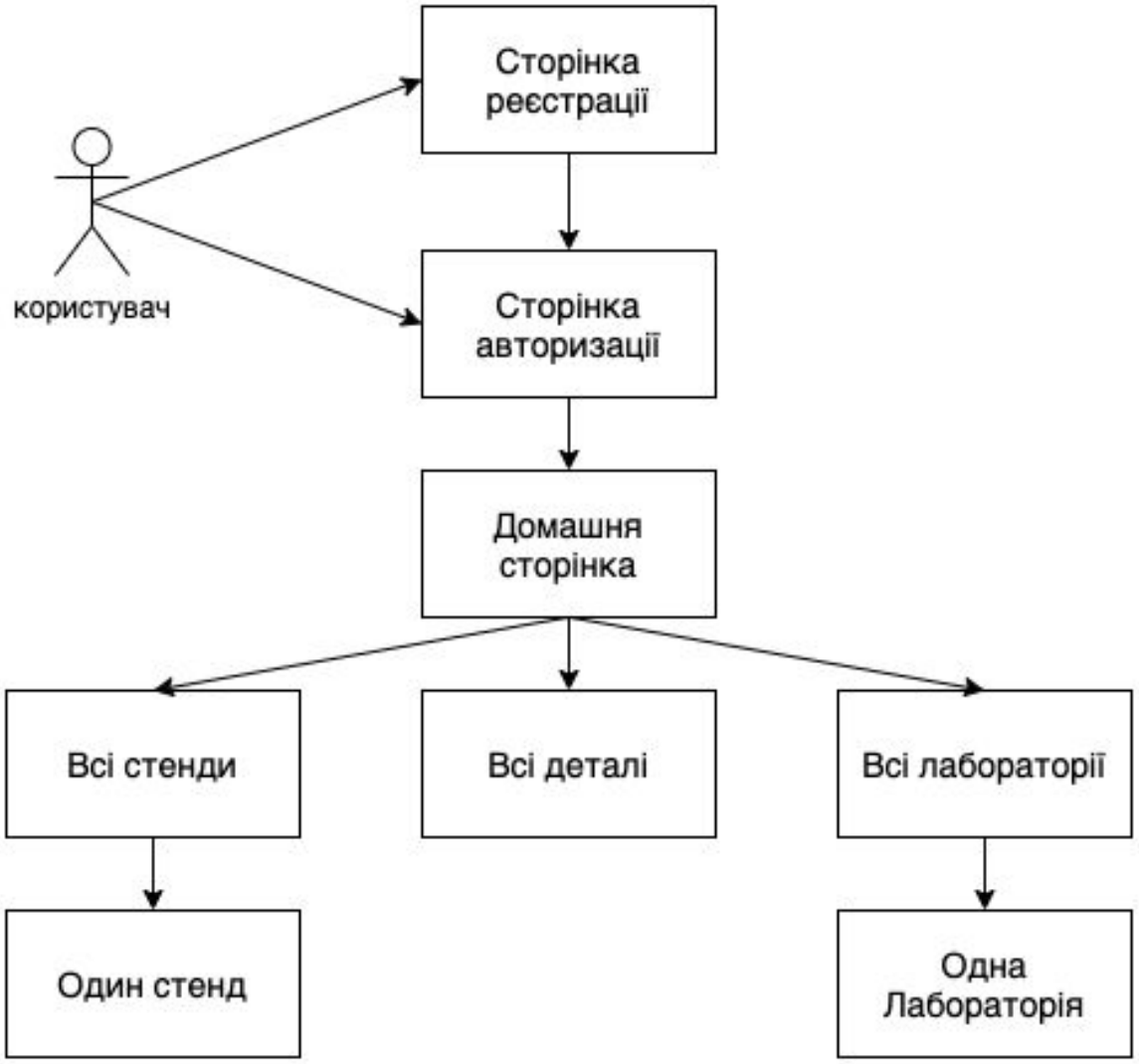

Рисунок 6.2 – діаграма поекранного відображення

На цій діаграмі зображена навігація користувача в середині системи. На першому екрані користувач може обрати реєстрацію або вхід в систему. Далі після входу користувач потрапляє на домашню сторінку. З неї користувач може перейти на сторінку усіх стендів, всіх деталей та всіх лабораторій. Зі сторінок усіх лабораторій та усіх стендів можна перейти на сторінку однієї лабораторії та одного стенду відповідно.

Кожен з елементів навігації в діаграмі поекранних переходів характеризується окремим посиланням. Єдиним виключенням є сторінка реєстрації та авторизації, вони працюють як перемикач між вкладками.

Розглянемо ще одну діаграму – діаграму класів. Класова діаграма[14] - це структурна діаграма UML, яка показує спроектовану системи на рівні класів та інтерфейсів, показує їх особливості, обмеження та зв’язки - асоціації, узагальнення, залежності тощо. Діаграма класів дає можливість показати логічну модель програми, на рівні класів. Вона надає статичне представлення про структуру системи – з яких класів вона складається, які зв’язки між цими класами, яких методів та полів складається кожний клас [13].

Розглянемо діаграму класів для серверної частини (рисунок 6.3). Спершу розглянемо класи Authenticable та Model. Клас Model є вбудованим класом фреймворка Laravel. Від нього наслідуються всі інші класи моделей. За рахунок такого наслідування кожен клас моделей отримує нові методи, зокрема методи для роботи з базою даних. Якщо створити модель Stand, а таблиця в базі даних матиме назву stands, то екземпляр класу Stand матиме простий доступ до до записів необхідної таблиці. Виключенням для моделей є модель User, яка наслідується від класу Authenticable, а не Model. Це пов’язано з тим, що користувач має функціонал реєстрації і авторизації. Для цього функціоналу “під капотом” фреймворка і створюються додаткові таблиці в базі даних.

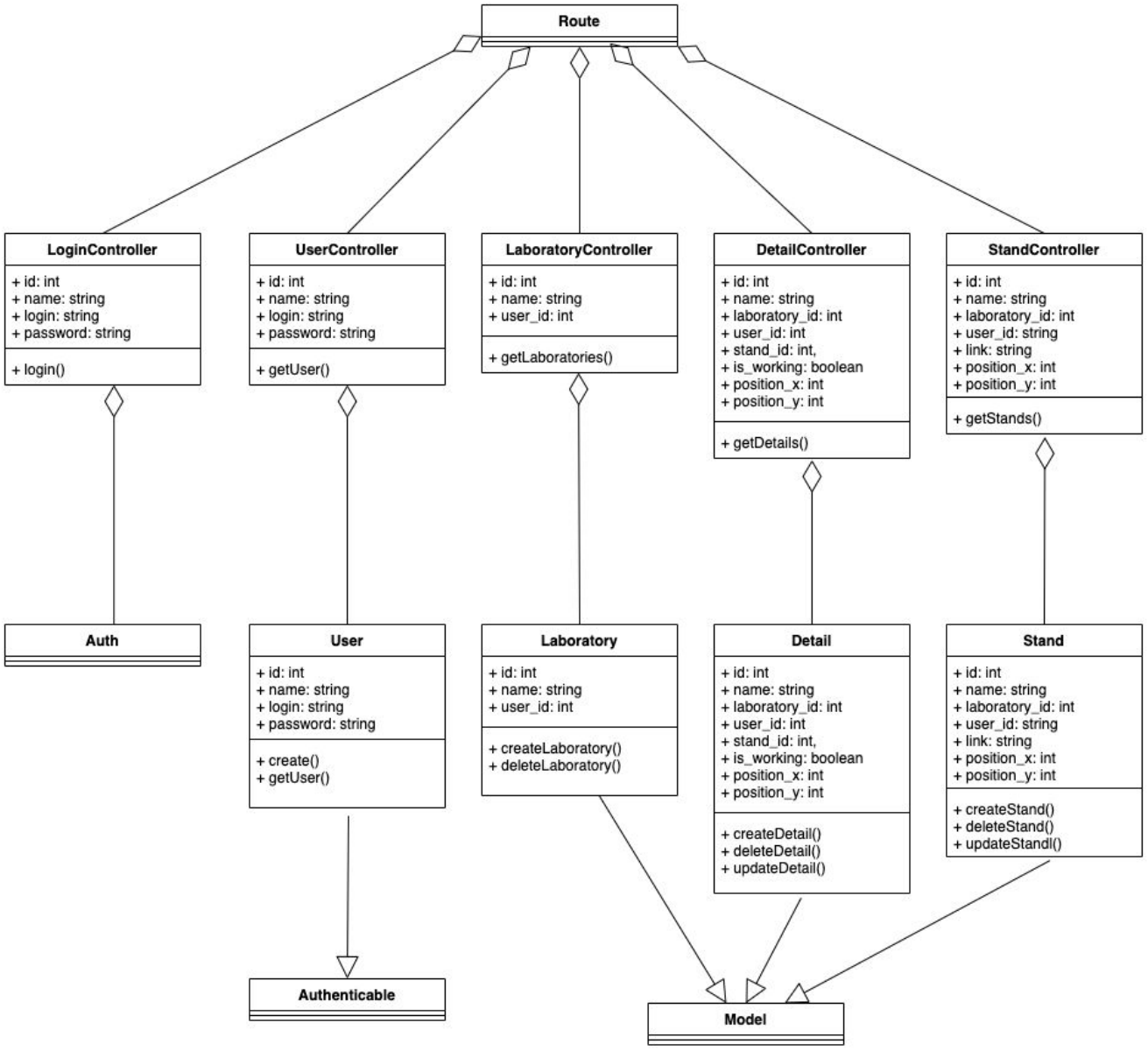

Рисунок 6.3 – діаграма класів серверної частини

Як вже було сказано, якщо назва класу моделі відповідає назві таблиці, проте в однині, то автоматично підключаються методи ORM. Такий клас отримує відкриті поля, назва та тип яких відповідають назві та типу полів що описані в під’єднаній таблиці. Тому всі наявні поля, які є в таблиці є полями самих таблиць, які описані детальніше в наступному розділі записки. Також однією з моделей є модель Auth, яка відповідає за авторизацію, і генерується системою автоматично.

Перейдемо до контроллерів. Кожен контроллер мусить агрегувати відповідну модель, оскільки йому необхідно викликати методи своєї моделі. Як

ми бачимо моделі містять основні методи роботи з сутностями, проте контроллерам можна делегувати прості операції, такі як отримання даних. Виключенням є хіба моменти, коли отримання даних супроводжується складнішою логікою, в такому випадку необхідно ці методи відділяти в модель. Єдиним контроллером який агрегує автоматично згенерований клас, а не створену вручну модель, є LoginController. Це пов’язано з особливостями авторизації.

Як результат – всі контроллери агрегуються в Router. Цей зв’язок не задається явно, а генерується інструментами Laravel через задану структуру застосунку.

Далі перейдемо до діаграми класів клієнтської частини. Vue.js насправді не використовує класичне визначення класів у тому вигляді, який пропонує мова програмування JavaScript, а використовує компонентний підхід. Кожен компонент складається з шаблону, JavaScript частини, та CSS стилів. Проте можна застосувати діаграму класів для компонентів, вважаючи стан компонента за його поля, за методи компонента за методи класу.

Діаграма класів клієнтської частини системи зображена на рисунку 6.4. На ній основновним класом, є сам клас Vue, який агрегує кореневий компонент, а також агрегує роутинг, сховище даних, та екземпляр класу Vuetify. З цих складових і створюється екземпляр класу Vue. Кореневий компонент App агрегує компонент MainNavigator, який відповідає за інтерфейс навігації всередині усієї системи. App включає в себе поля і методи для роботи з токеном, оскільки вся робота системи грунтується на тому, щоб користувач був авторизований.

Клас Store відображає глобальний стан системи. Він містить в собі сам стан, та об’єкти які включають методи для роботи з цими даними, та асинхронними операціями.

Клас Vuetify відповідає за інтерфейс, та елементарні будівельні блоки нашої системи.

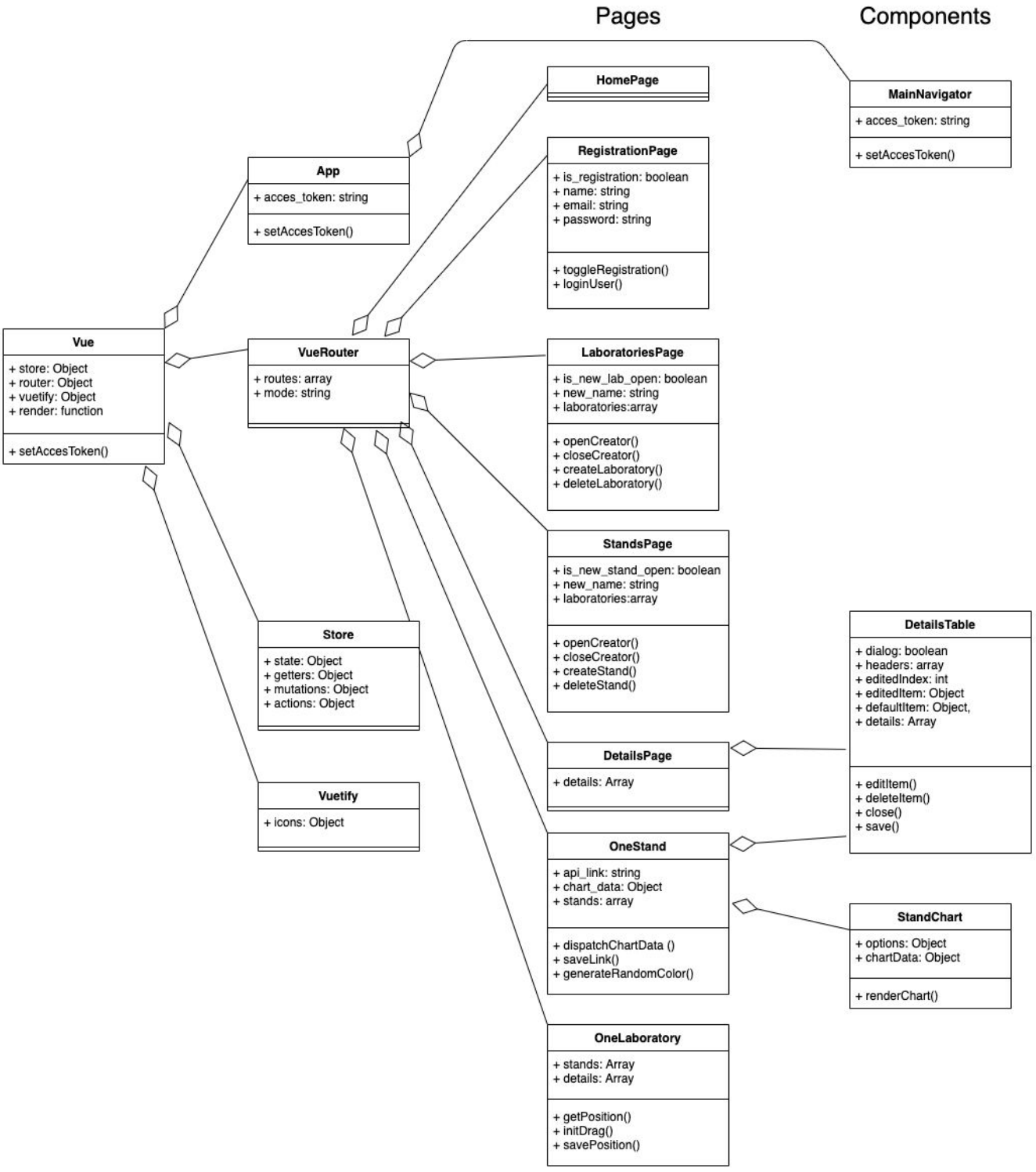

Рисунок 6.4 – діаграма класів клієнтської частини

VueRouter агрегує в себе усі сторінки. Сторінки – це компоненти, кожен з яких представлений своїм унікальним посиланням в системі. Всього є 7 унікальних сторінок, вони розташовані на діаграмі під заголовком Pages. Кожна сторінка в своєму стані містить елементи які необхідні для відображення на

інтерфейсі користувача, а також методи взаємодії користувача з інтерфейсом та обробки даних.

1. Компонент HomePage є статичною сторінкою, яка служить для навігації на інші сторінки, тому не має лише один стан, та не має методів.
2. Компонент RegistrationPage може бути представлений двома станами: формою реєстрації та формою авторизації. За ці стани відповідає поле is_registration. Інші поля прив’язані до значень полів, а методи відповідають за керування станом та відправлення даних на сервер.
3. Третій компонент LaboratoriesPage відповідає за інтерфейс усіх лабораторій. У ньому представлені списком усі лабораторії, а його поля відповідають за стан відображення попапа для створення нової лабораторії.
4. Компонент StandsPage відповідає за інтерфейс усіх стендів, та є за своєю структурою схожим до структури компоненту LaboratoriesPage. У ньому представлені списком усі стенди, перше вказане поле типу boolean відповідає за стан попапа для додавання нових стендів. Також присутні методи для видалення стендів.
5. П’ятий компонент – DetailsPage. Він включає в себе лише одне поле details, яке передає далі в компонент DetailsTable, який в свою чергу реалізує всю необхідну для роботи з таблицями логіку.
6. Компонент OneStand, як і DetailsPage агрегує компонент DetailsTable, для реалізації роботи з таблицями. Також компонент OneStand агрегує компонент StandChart для відмальовування графіків. Для цього він має методи для роботи з підключенням стороннього API, а також для генерації кольорів створених графіків.
7. Сьомий компонент – OneLaboratory. Він включає в себе поля стендів, та деталей, наявних в конкретній лабораторії. Також містить методи для отримання позиції, збереження позиції, та ініціалізації методів для перетягування об’єктів.

Останній два компоненти – DetailsTable та StandChart. DetailsTable використовується для відображення таблиць деталей на сторінках одного стенду та сторінці всіх деталей. Цей компонент надає весь функціонал роботи з таблицями – перегляд, додавання, видалення, редагування, сортування і т.д.. Компонент StandChart агрегується компонентом OneStand і отримує в якості вхідних параметрів дані про стенд, та згенеровані кольори графіків. На їх основі він і малює графіки з використанням технології canvas.

7. Опис програмної реалізації

Опис програмної реалізації можна розділити на три частини: клієнтська, серверна і база даних.

Фреймворк Laravel і Vue.js заохочують нас до використання двох патернів: MVC (model–view–controller) на серверній частині, та MVVM (Model-View-ViewModel) на клієнтській. Зв’язок між клієнтською і серверною частиною реалізується через REST API (рисунок 7.1).

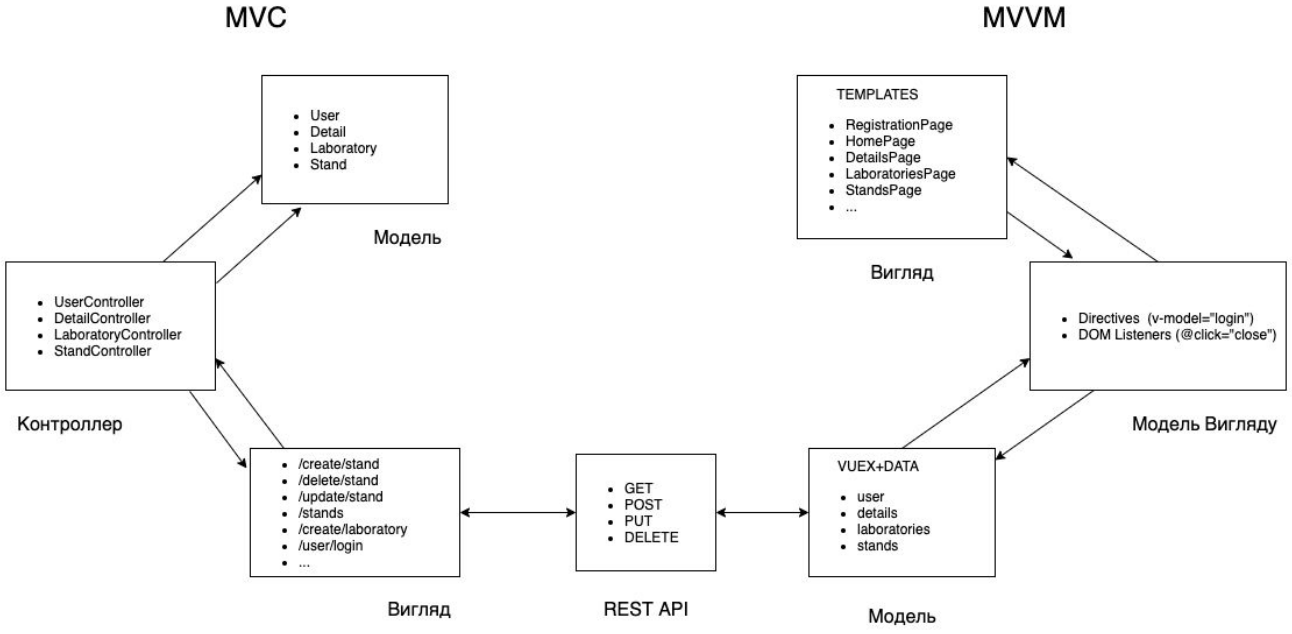

Рисунок 7.1 – структура проекта

Модель на сервері представляє основу бізнес логіки та надає нам доступ через ORM до бази даних. Далі контроллер викликає методи нашої моделі і передає результат на вигляд, який в даному випадку виступає шляхами зв’язку з клієнтською частиною. Аналогічно працює зворотній зв’язок, тільки між видом і контроллером існує ще прошарок middleware, який перевіряє чи прийшов запит від авторизованого користувача.

Що стосується клієнтської частини, то у неї модель представлена у якості сховища даних Vuex та станом окремих компонентів data. Прив’язка усіх даних до

вигляду відбувається через директиви - властивості які надає нам фреймворк. Зворотній зв’язок від вигляду можна отримати також через слухачі подій. Директиви і слухачі подій і є власне представниками моделі вигляду.

На клієнтській частині ми також використовуємо localStorage - сховище браузера для зберігання токену користувача, на серверній ми використовуємо базу даних MySQL.

7.1. Клієнтська частина

Клієнтська частини застосунку була розроблена з використанням фреймворку Vue.js, а також плагіну Vuetify, який значно полегшує структуризацію даних на стороні клієнта. Для розробки був використаний паттерн MVVM. Технічно сам Vue.js орієнтований на шар ViewModel шаблону MVVM. Він з'єднує View і Model за допомогою двосторонніх прив'язок даних. Фактичні маніпуляції з DOM та форматування вихідних даних віднесені до директив та фільтрів (рисунок 7.1).

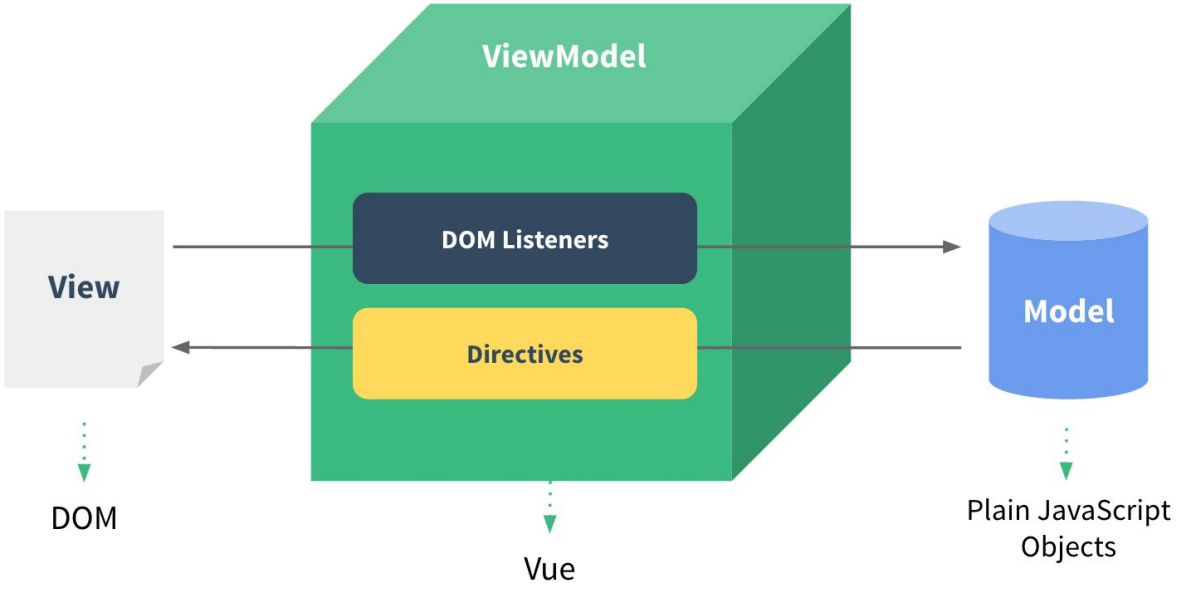

Рисунок 7.1 — MVVM з використанням Vue.js

Шаблон MVVM ділиться на три частини:

Модель (Model), як і в класичному шаблоні MVC, Модель являє собою фундаментальні дані, що необхідні для роботи застосунку. В розробленому застосунку модель представлена сховищем даних Vuex[3] рисунок 7.2, яке зберігає поточний стан застосунку.

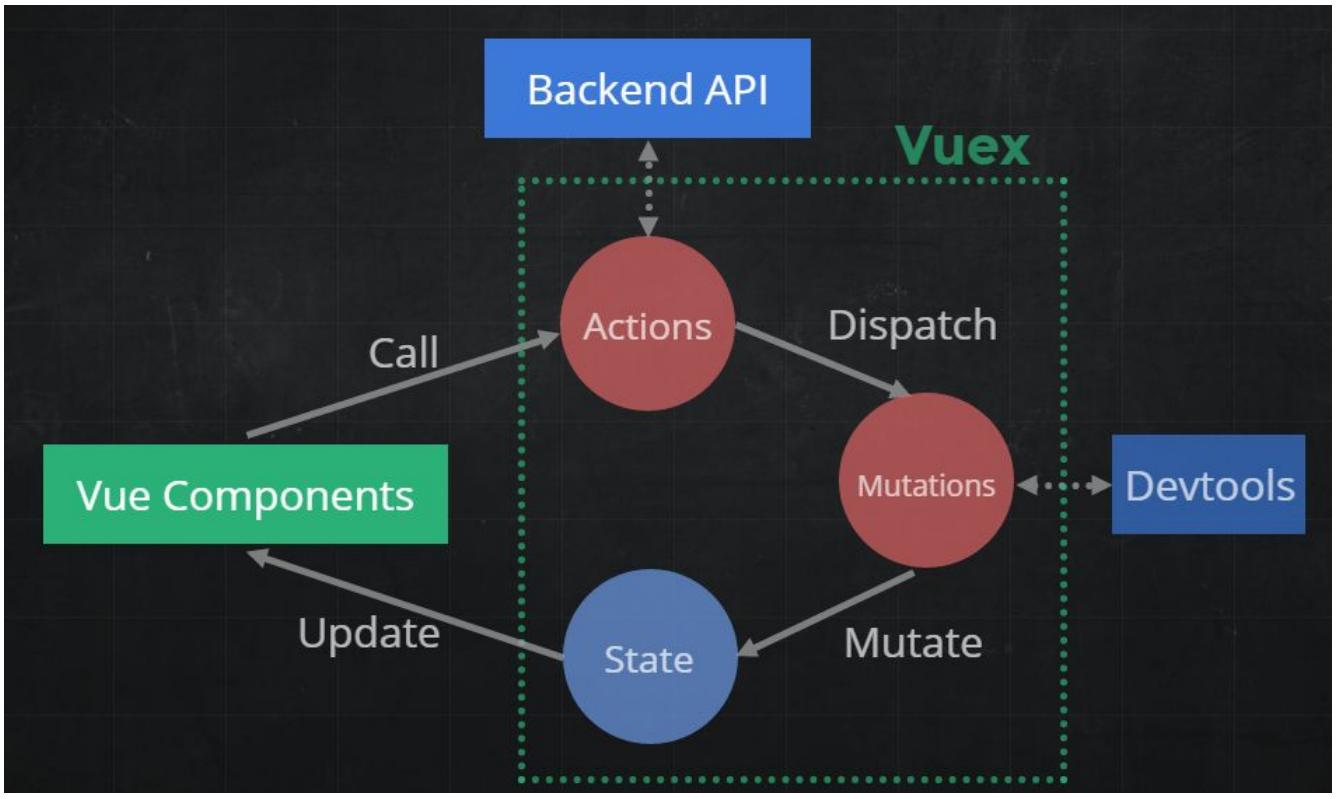

Рисунок 7.2 – схема роботи Vuex

Два моменти відрізняють сховище Vuex від простого глобального об'єкта:

1. Сховище Vuex — реактивне. Коли компоненти Vue покладаються на його стан, то вони будуть реактивно і ефективно оновлюватися, якщо стан сховища змінюється.
2. Не можна безпосередньо змінювати стан сховища. Єдиний спосіб внести зміни — явно викликати мутацію. Це гарантує, що будь-яка зміна стану залишає слід і дозволяє використовувати інструментарій, щоб краще розуміти хід роботи програми.

Vuex використовує єдине дерево стану - коли один об'єкт є єдиним глобальним станом застосунку. Для великих проектів стан сховища поділяють на модулі, проте в даному випадку такої необхідності немає. В розробленому застосунку було виділено основні його сутності, а саме - користувач, стенди, лабораторії та деталі. Сутність користувача необхідна для зв’язку з серверною частиною. Ця сутність є типом даних об’єкт. Решта перелічених сутностей - це масиви даних. Для щойно зареєстрованого користувача, масиви стендів і деталей будуть пустими, проте масив лабораторій буде містити одну лабораторію, яка створюється за замовчуванням. Це зроблено з логічних міркувань оскільки кожен стенд чи деталь повинна мати своє місце знаходження.

Друга складова сховища - це геттери. Інколи може виникати потреба вирахувати стан застосунку на основі стану сховища. Наприклад відфільтрувати список, а потім повернути кількість його елементів. В такому випадку використовується спеціальна властивість - геттери. В розроблені системі геттери використовуються щоб отримати список ідентифікаторів лабораторій та стендів при створенні та редагуванні деталей. Цей список виводиться у полі типу select і його клієнт може побачити як випадаючий список.

Єдиним способом змінити стан у сховища Vuex – використати мутації. Мутації – це функції, які приймають першим параметром стан сховища, яке вони мають змінити, а другим параметром – дані, які необхідні для присвоєння нового стану (другий параметр не є обов’язковим). Простіше кажучи, мутації є так званими сеттерами. Більшість мутації виконують просту функцію - замінюють елемент поточного стану сховища на новий отриманий елемент. Для таких змін в розробленій системі була створена мутація mutate, яка отримує в якості другого параметра об’єкт, що містить ключ – назву змінної стану сховища, і нове значення. Проте для випадків, коли дані які необхідно записати до стану сховища, потребують додаткової обробки, необхідно створювати і викликати окремі мутації.

Остання складова сховища – це дії. Дії - схожі на мутації, проте з декількома відмінностями:

1. Замість того, щоб змінити стан, дії повинні ініціювати мутації.
2. Дії можуть бути використані для асинхронних операцій.

Другий пункт грає ключову роль, оскільки саме через дії будуть відбуватися запити на серверну частину застосунку. В першу чергу це будуть запити типу GET, для отримання даних з серверної частини. Прикладом такої дії в створеному застосунку є дія getStands, яка буде отримувати всі стенди користувача і передавати їх в необхідну мутацію. Проте можуть бути і запити типу POST. Прикладом такої мутації є мутація loginUser. Вона буде повертати токен авторизації, який буде використовуватись в подальшому для всіх запитів що стосуються авторизованого користувача.

Вид/(Вигляд) (View) як і в класичному шаблоні MVC, Вигляд — це графічний інтерфейс, тобто вікно, кнопки тощо, які відображені у шаблонах. Vue.js використовує шаблони на основі DOM. Кожен екземпляр Vue асоціюється з відповідним елементом DOM. Коли створюється екземпляр Vue, він рекурсивно обходить усі дочірні вузли його кореневого елемента, встановлюючи необхідні прив’язки даних. Після складання View, він стає реактивним на зміни даних. В розробленому застосунку використовується плагін Vuetify, тому шаблон в більшості містить вже готові компоненти, і рідше — окремі DOM елементи. В самому.

Модель вигляду (ViewModel, що означає «Model of View»[1]) з одного боку є абстракцією Вигляду, а з іншого надає обгортку даних з Моделі, які мають зв'язуватись. Тобто вона містить Модель, яка перетворена до Вигляду, а також містить у собі команди, якими може скористатися Вигляд для впливу на Модель. Фактично ViewModel призначена для того, щоб:

1. Здійснювати зв'язок між моделлю та відображенням.

2. Відслідковувати зміни в даних, що зроблені користувачем.
3. Відпрацьовувати логіку роботи View (механізм команд).

Як вже було сказано вище, при використанні Vue.js, роль ViewModel дістається директивам та фільтрам.

У багатьох проектах, глобальні компоненти визначають Vue.component, з кожним новим Vue ({el: '#container'}) для вказівки елементів-контейнерів в тілі кожній сторінці. Для проектів малого та середнього розмірів, у яких JavaScript використовується для деяких сторінок, цей проект може прекрасно працювати. У більш складних проектах або у випадках, коли ваш власний фронтенд управляє JavaScript, є деякі недоліки:

1. Глобальне визначення змушує давати унікальне ім'я кожному компоненту.
2. Шаблоном не вистачає підсвітки синтаксису.
3. Немає модульної підтримки CSS — у той час як HTML і JavaScript розбиваються на модулі-компоненти, CSS опиняється за бортом.
4. Відсутність складальника проекта обмежує нас чистим HTML та JavaScript ES5, і не дозволяє використовувати перепроцессори (наприклад Pug або Babel).

Всі ці недоліки вирішуються з допомогою однофайлового компонента[6] з розширенням “.vue”, використання яких, дозволяють нам такі інструменти як Webpack і Browserify. Саме тому в розробленій системі використовується підхід однофайлових компонентів, де для кожного компонента в одному файлі ми отримуємо одразу розмітку елементів, JavaScript код, і CSS стилі.

7.2. Серверна частина

Серверна частина була реалізована з використанням фреймворку Laravel та застосуванням паттерну MVC.

У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами (рисунок 7.3).

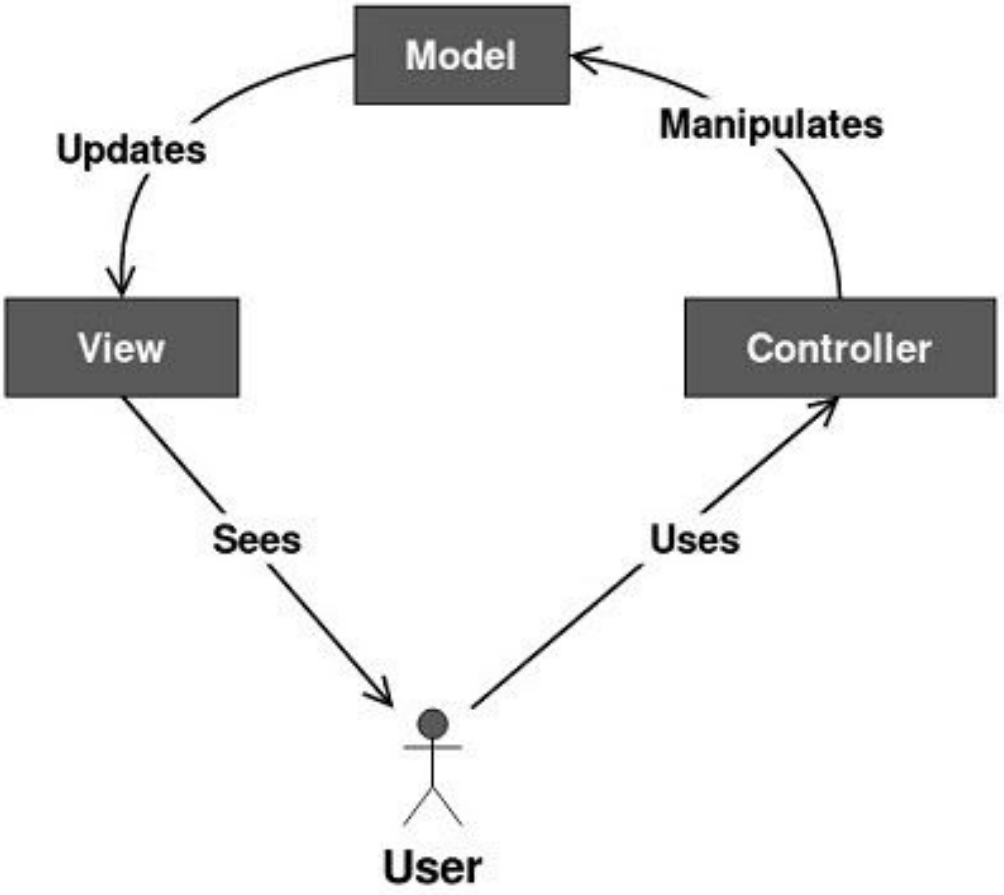

Рисунок 7.3. – схема роботи MVC

- Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.
- Вигляд може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії.
- Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

Laravel — один з найбільш поширених PHP фреймворків для веб-розробки. Він відповідає архітектурній схемі MVC (Model-View-Controller), та надає деякі дивовижні функції для розробки Laravel, такі як повна система аутентифікації, потужна ORM, міграція баз даних, пагінація та багато іншого. Оскільки в нас більшість маніпуляцій з даними знаходиться на стороні клієнта, то ми використовуємо фреймворк Laravel для написання REST API (рисунок 7.4). Вся взаємодія з сервером зводиться до 4 операцій :

1. Отримання даних з сервера (у форматі JSON).
2. Додавання нових даних на сервер.
3. Модифікація існуючих даних на сервері.
4. Видалення даних на сервері.

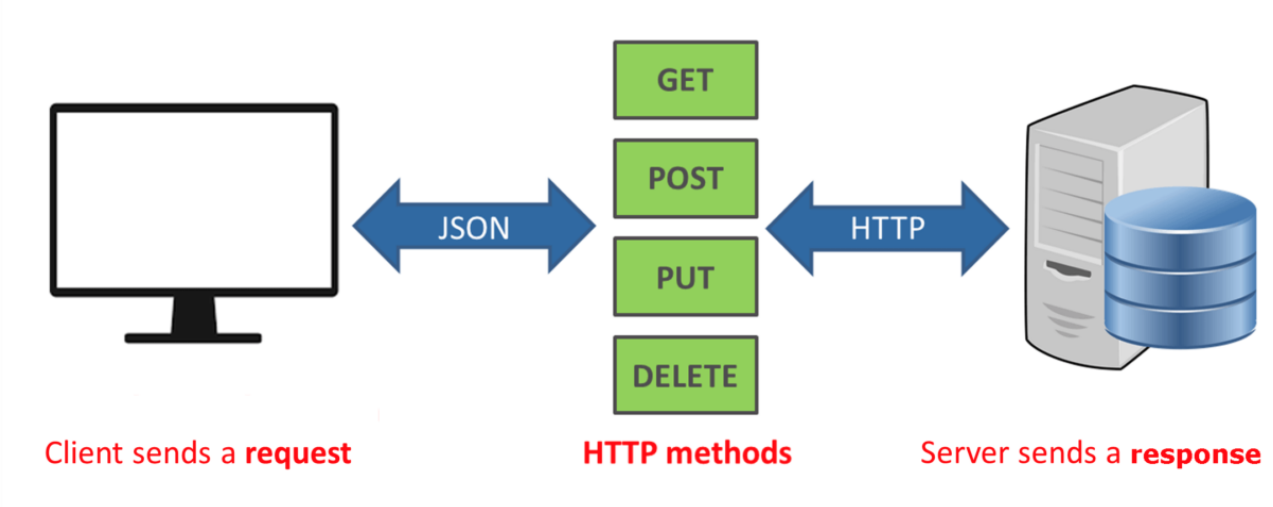

Рисунок 7.4– принцип роботи REST API

Для кожного типу операції використовується свій метод HTTP-запиту:

1. Отримання — GET.
2. Додавання — POST.
3. Модифікація — PUT.
4. Видалення — DELETE.

Всі HTTP-запити описані в одному файлі. Далі кожен запит викликає відповідний контроллер, який в свою чергу звертається до необхідних методів моделі.

7.3. База данних

Баз даних була розроблена з використанням MySQL — вільної система керування реляційними базами даних. Схема бази даних зображена на рисунку 7.5.

Було створено 4 таблиці: користувачі, лабораторії, стенди та деталі. Усі зв’язки між таблицями ідуть за принципом “один до багатьох”. Також у кожної таблиці є свої поля created_at та updated_at які визначаються типом даних timestamp. Використовуючи ORM від фреймворка Laravel ці поля будуть заповнюватись автоматично.

Таблиця користувачів генерується автоматично, оскільки при розробці програмного продукту використовуються інструменти фреймворка Laravel для реалізації реєстрації та авторизації. Проте таблицю можна редагувати і доповнювати своїми власними полями.

Кожен користувач може мати багато лабораторій, за замовчуванням буде створена хоча б одна лабораторія. Звісно, користувач може видалити її, в такому випадку вона буде визначена типом null.

Далі кожна лабораторія може мати багато стендів. Кожен стенд може мати свою позицію у лабораторії, яка визначається двома координатами: position_x та position_y. Ці координати визначені типами tinyint, оскільки місце розташування найзручніше визначати у відсотках – від 0 до 100%.

Розглянемо таблицю деталей. У випадку якщо у нас немає стендів, а на даний момент ми хочемо лише вести облік деталей, то поле stand_id яке служить зовнішнім ключем може також набувати значення null. Так само як і стенди, деталям можна задати позицію в лабораторії, саме тому створений зв’язок між таблицею лабораторій таблицею деталей. Також аналогічно до стендів для

деталей задаються координати position_x та position_y. Кожен стенд може містити багато деталей.

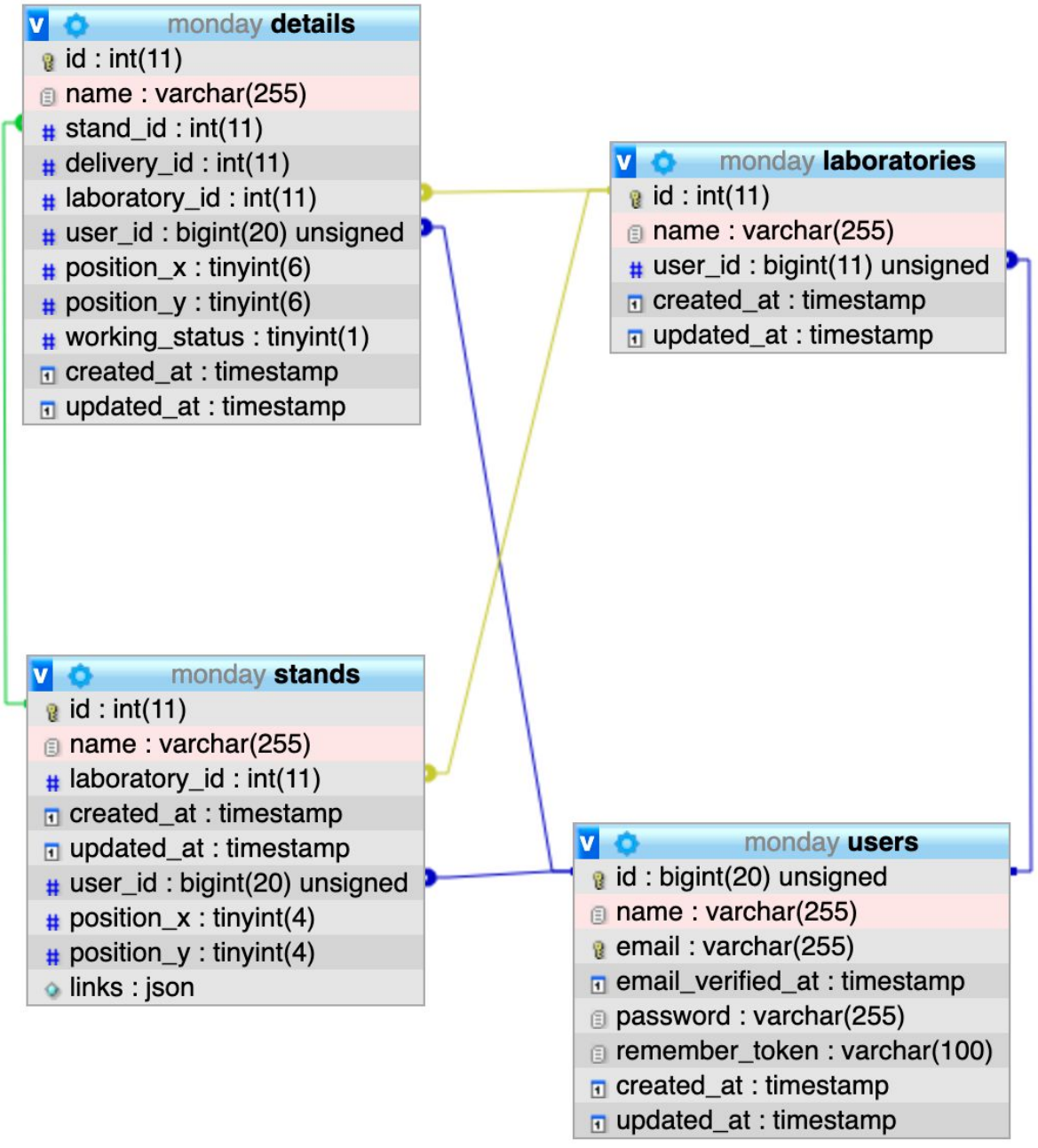

Рисунок 7.5. – схема бази даних

Для створення зв’язків між таблицями необхідно з одного боку вказати унікальний ключ – primary key, а з іншого боку вказати поле яке буде служити в якості зв’язку з іншою таблицею — foreign key. У якості primary key у кожної таблиці створено свій унікальний id. Primary key визначений типом int, і є

автоінкрементом. Варто зауважити, що кожна з сутностей має свій зв’язок із таблицею користувача, адже кожна сутність прикріплена до користувача, і інші користувачі не повинні отримувати до неї доступ. Також на відміну від інших зв’язків між таблицями, у зв’язків з ьаблицею користувача неможливо встановлювати значення поля null, оскільки сутності деталей, стендів та лабораторій не можуть існувати окремо від користувача.

Для реалізації проекту створюються таблиці, які не стосуються загальної схеми, і які не мають явних зв’язків. Прикладами таких таблиць є таблиця токенів (рисунок 7.6) і таблиця міграцій (рисунок 7.7).

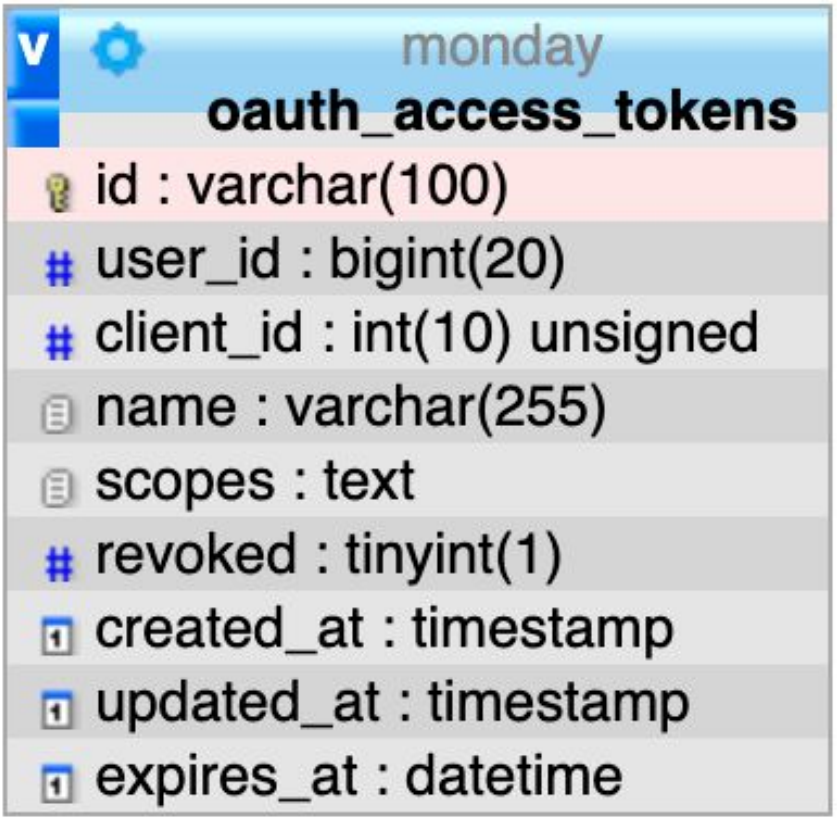

Рисунок 7.6. – Таблиця міграцій

Таблиця токенів генерується автоматично засобами фреймворка. Вона містить ідентифікатор користувача, який здійснює вхід в систему, саме значення

токена, а також, окрім стандартних полів created_at та updated_at ще одне поле expires_at. Це поле визначає час “протухання” токена, тобто його термін придатності. Після закінчення терміну придатності токена він стає недійсним, і користувачу необхідно буде виконати повторний вхід в систему. За замовчуванням час виходу терміну придатності токена становить 1 рік, проте звісно цей час можна редагувати, якщо є така необхідність.

Ще одна таблиця яку варто розглянути, це таблиця міграцій (рисунок 7.7).

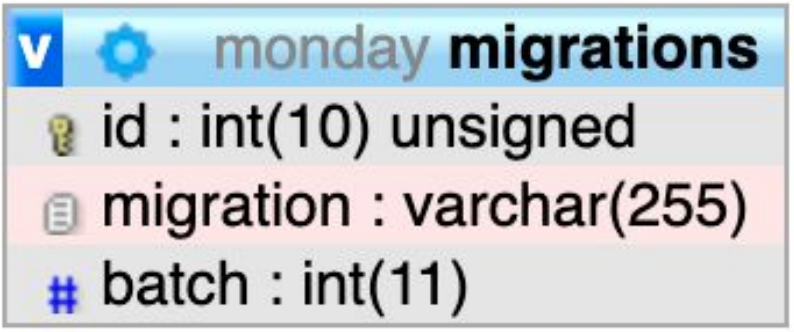

Рисунок 7.7. – Таблиця міграцій

Таблиця міграцій необхідна для відслідковування усіх змін які стосуються самої структури баз даних. Вона не є частиною системи, проте використовується для спрощення розробки.

Існують і інші таблиці які генеруються автоматично, проте вони не є частиною предметної області, тому їх розглядати не будемо.

5.4. Взаємодія клієнт – сервер

Вся взаємодія клієнт – сервер відбувається через REST API, і має деякі особливості у своїй структурі, які стосуються в першу чергу необхідності авторизації користувача. Для створення запитів зі сторони клієнта використовується бібліотека axios, яка є легкою, та базується на XMLHttpRequests сервісі. В розробленій системі моніторингу ця бібліотека використовується з базовими налаштуваннями лише для двох видів запитів. Перший запит - це запит

для реєстрації користувача, а другий - для його авторизації. Всі інші запити працюють з уже авторизованим користувачем, тому для них необхідно додавати специфічний заголовок Authorization. У якості значення для цього заголовку прикріплюється токен, який був раніше згенерований сервером.

Важливим моментом, є збереження токену після перезавантаження сторінки. У випадку коли авторизація реалізована через сесії сервера, ця проблема зникає, оскільки вся інформація міститься в куки. Проте в даному випадку треба зберігати токен вручну через javascript. Для такого випадку можна розглянути 2 види сховищ - sessionStorage і localStorage. Вони дозволяють зберігати пари ключ / значення в браузері, і мають ряд відмінностей від куки.

1. На відміну від куки, об'єкти веб-сховища не потрапляють на сервер при кожному запиті. Тому ми можемо зберігати набагато більше даних. Більшість браузерів можуть зберегти як мінімум 2 мегабайти даних (або більше), і цей розмір можна поміняти в налаштуваннях.
2. Ще одна відмінність від куки - сервер не може маніпулювати об'єктами сховища через HTTP-заголовки. Все робиться за допомогою JavaScript.
3. Сховище прив'язане до джерела (домен / протокол / порт). Це означає, що різні протоколи або піддомени визначають різні об'єкти сховища, і вони не можуть отримати доступ до даних один одного.

Основні особливості localStorage:

- Цей об'єкт один на всі вкладки та вікна в рамках джерела (один і той же домен / протокол / порт).
- Дані не мають терміну давності, за яким закінчуються і видаляються. Зберігаються після перезапуску браузера і навіть ОС.

В свою чергу властивості sessionStorage мають деякі відмінності від localStorage:

- sessionStorage існує тільки в рамках поточної вкладки браузера. Інша вкладка з тієї ж сторінкою матиме інше сховище.

- Дані продовжують існувати після перезавантаження сторінки, але не після закриття / відкриття вкладки.

Порівнявши два види сховищ браузерів стає зрозуміло, що для авторизації токен найзручніше буде зберігати в localStorage, щоб користувач міг спокійно змінювати вкладки браузера і при тому залишати статус входу в систему. Записані в localStorage та sessionStorage дані можна завжди переглянути в панелі розробника браузера (рисунок 7.8)

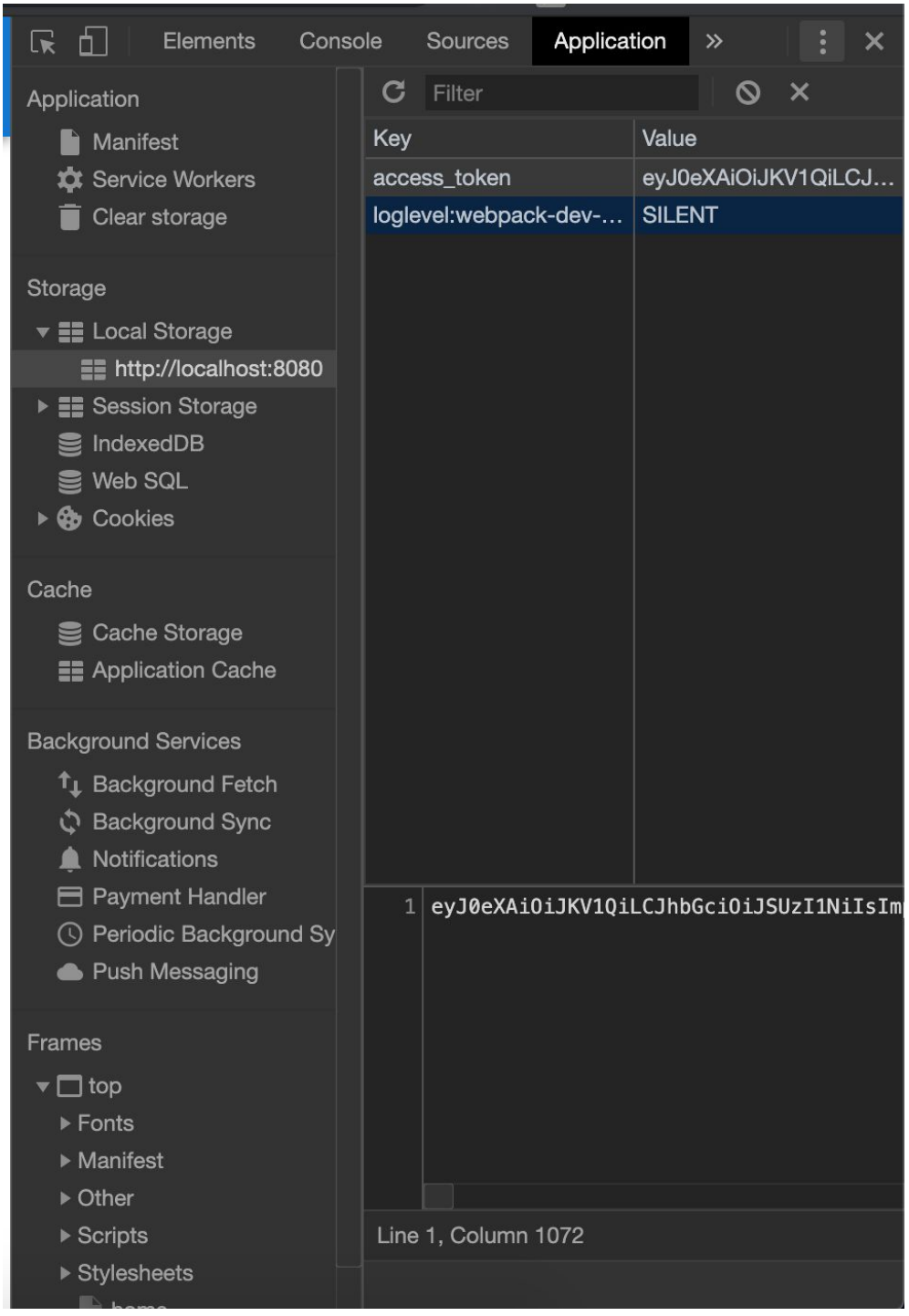

Рисунок 7.8 - дані записані в localStorage

Після додавання токена в localStorage його можна завжди звідти дістати, проте насправді незручно це робити кожного разу при здійсненні запитів. Для цього сам токен буде діставатись лише один раз – при першому запуску застосунку. Далі для всіх наступних разів реалізується надбудова над бібліотекою axios. Ця надбудова виконує всі ті самі функції що і сама бібліотека, тільки для неї попередньо прописані два заголовки – Authorization та Content-type. Перший заголовок містить токен та повідомляє серверу, що запит іде від авторизованого користувача. Другий заголовок повідомляє серверу про тип в якому будуть передаватись дані, в нашому випадку це multipart/form-data.

Також для спрощення коду та зменшення його кількості був створений ще один плагін, який відповідає за приведення даних, що надсилаються до потрібного вигляду. XMLHttpRequest дає підтримку для інтерфейсу FormData[8]. Об'єкти FormData дозволяють легко конструювати набори пар ключ-значення, що представляють поля форми і їх значення, які в подальшому можна відправити з використанням javaScript методів. Для реалізації цього плагіну був застосований паттерн проектування builder. Builder (Будівельник) — це породжувальний патерн проектування, що дає змогу створювати об’єкти крок за кроком. Будівельник дає можливість використовувати один і той самий код будівництва для отримання різних відображень об’єктів. Патерн Будівельник пропонує винести конструювання об’єкта за межі його власного класу, доручивши цю справу окремим об’єктам, які називаються будівельниками.

В даному випадку нам необхідно створити екземпляр класу FormData. Ми не можемо створити цей екземпляр передавши просто об’єкт з даними. Для цього необхідно додавати кожне значення окремо, використовуючи метод append. Для того щоб не робити це все вручну була створена функція formData Builder. Вона приймає у якості вхідних параметрів об’єкт даних (які в даному випадку і виконують роль будівельників) і повертає вже готовий екземпляр класу FormData.

Для сервера також важлива якого саме виду здійснюється запит. Для обробки запитів авторизованого користувача на сервері необхідно викликати middleware, який ідентифікує чи присутній в заголовку попередньо виданий токен. Якщо він присутній, то далі сервер може успішно продовжувати свою роботу, інакше, сервер поверне помилку зі статусом 401. Для роботи з авторизованим користувачем на сервері стає доступний клас Auth, який містить методи, що полегшують роботу з даними. Зокрема через цей клас можна швидко доступитись до об’єкта користувача з бази даних, або його унікального id.

8. Робота користувача з програмною системою

Інтерфейс користувача був реалізований з допомогою плагіна Vuetify, тому він простий у використанні і виконаний за принципом матеріального дизайну (Google Material Design). Матеріальний дизайн — принципи дизайну сайтів, програмного забезпечення і застосунків, а також правила дизайну інтерфейсів для операційної системи Android від компанії Google. Вперше представлений на конференції Google I/O 25 червня 2014 року. Ідея дизайну полягає в інтерфейсі, поведінка і вигляд якого наслідують правила поведінки і вигляду паперових карток в реальному житті.

Розглянемо поекранне використання веб-системою. При першому відкритті системи користувач має або зареєструватися (рисунок 8.1).

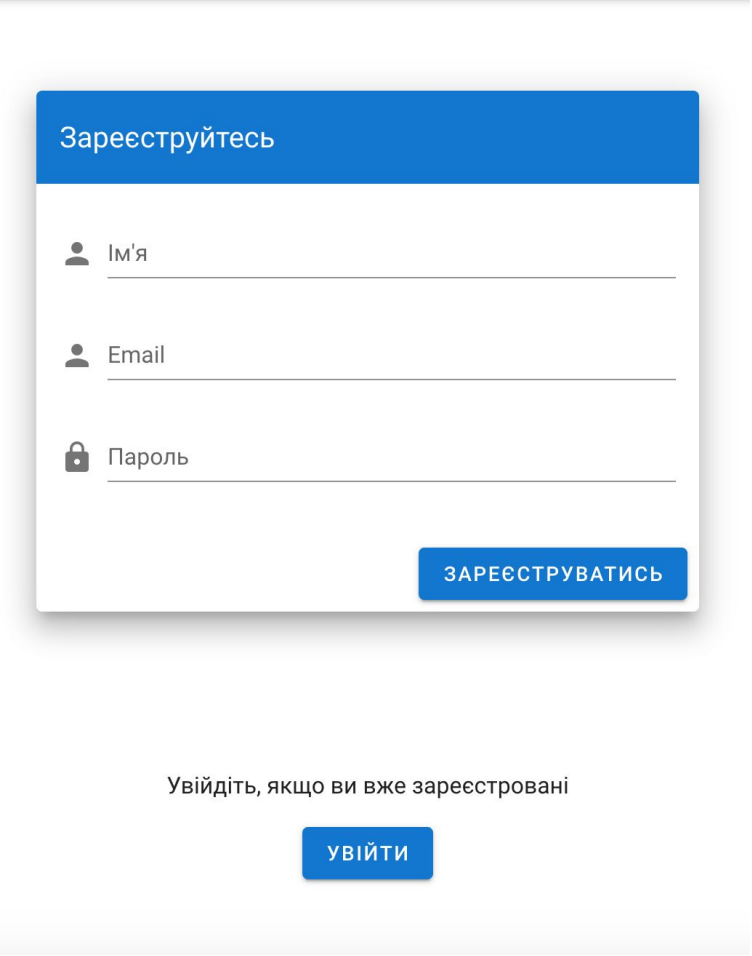

Рисунок 8.1— Реєстрація користувача

Якщо користувач вже зареєстрований, то йому необхідно увійти в систему.

Інтерфейс входу в систему (рисунок 8.2) виглядає подібно до інтерфейсу реєстрації .

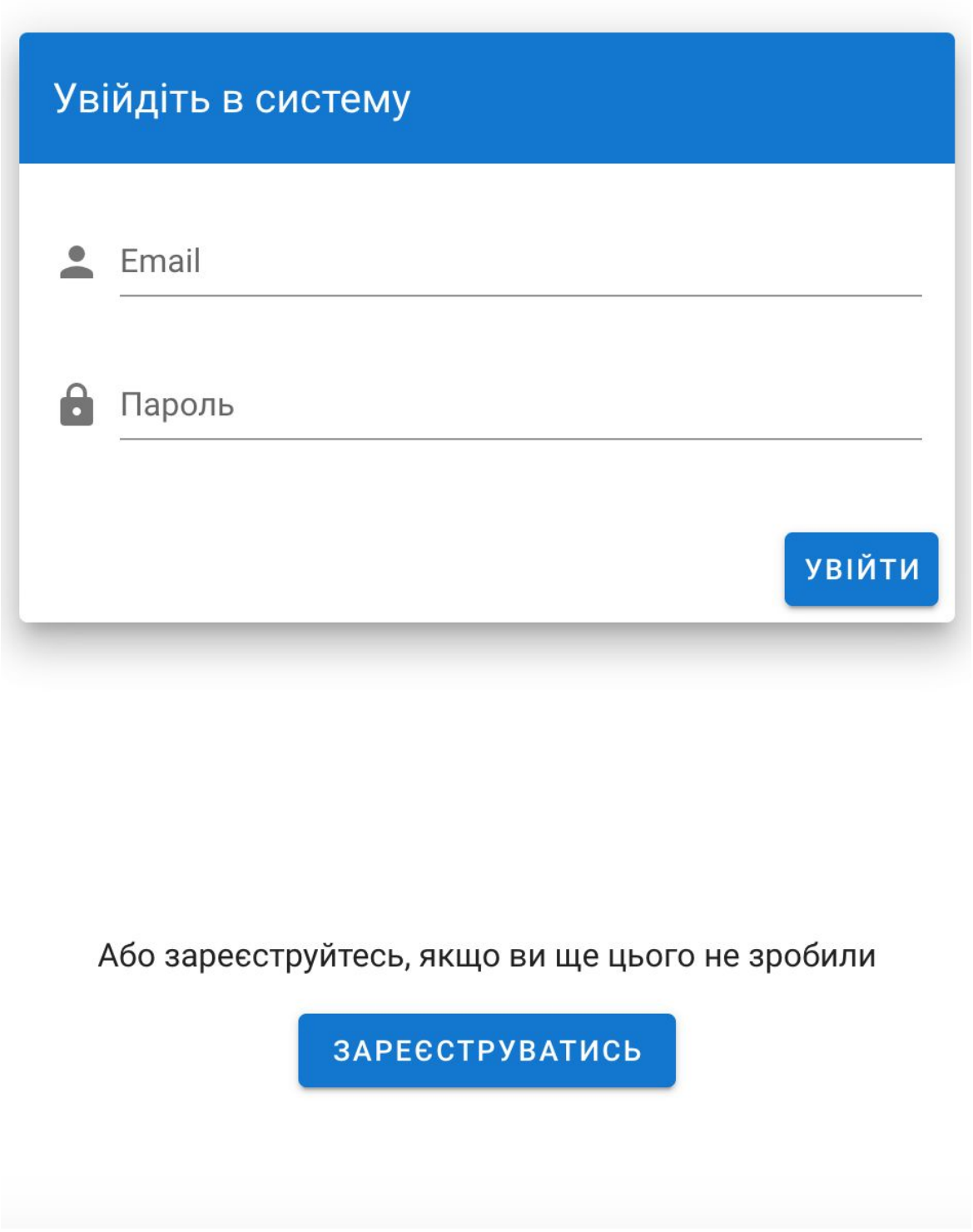

Рисунок 8.2— Авторизація користувача

Користувач повинен ввести свій логін та пароль, і увійти в систему. На стороні клієнта працює валідація, яка дозволяє надсилати запити на сервер лише у випадку коректних даних, інакше система повідомляє про помилку (рисунок 8.3).

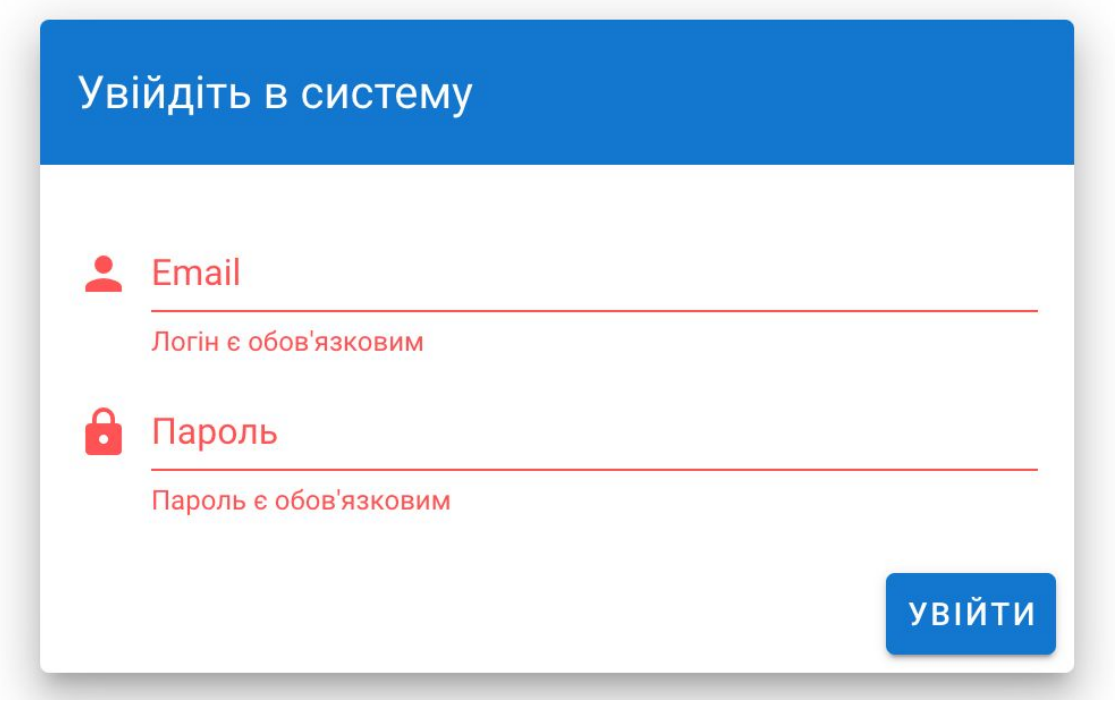

Рисунок 8.3— Повідомлення про некоректні дані

Після успішного входу в систему користувач переходить на домашню сторінку (рисунок 8.4) і може обрати необхідний йому функціонал.

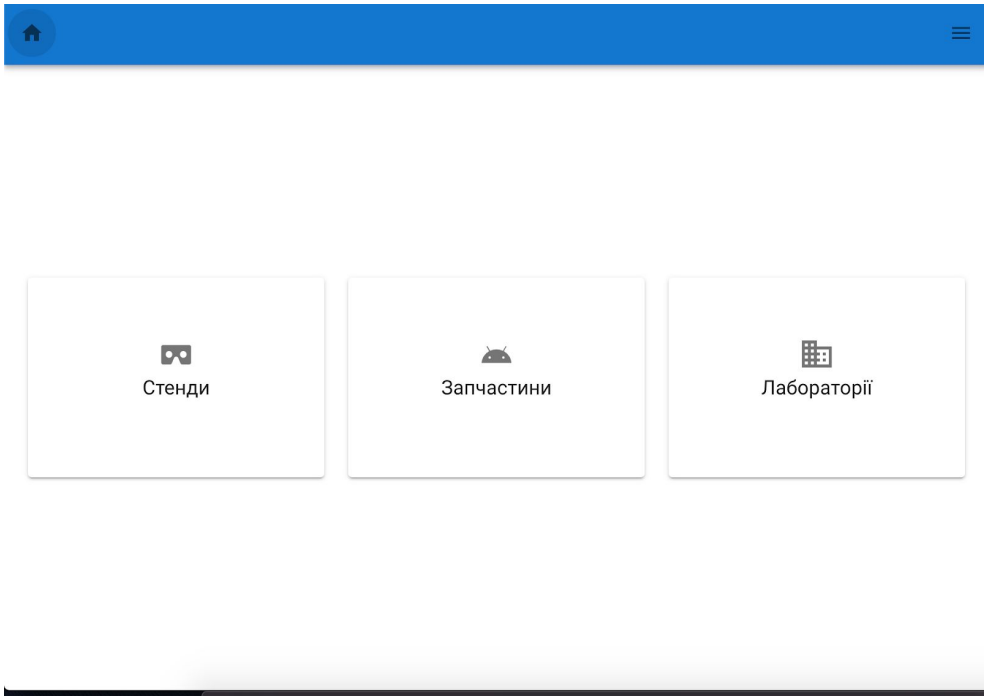

Рисунок 8.4 — Домашня сторінка

Також переглянути весь доступний функціонал, і обрати потрібний можна з випадаючого меню (рисунок 8.5).

Для того щоб його переглянути, необхідно натиснути на значок у верхньому правому кутку екрана. Також з випадаючого меню доступне посилання на домашню сторінку та кнопка виходу з системи.

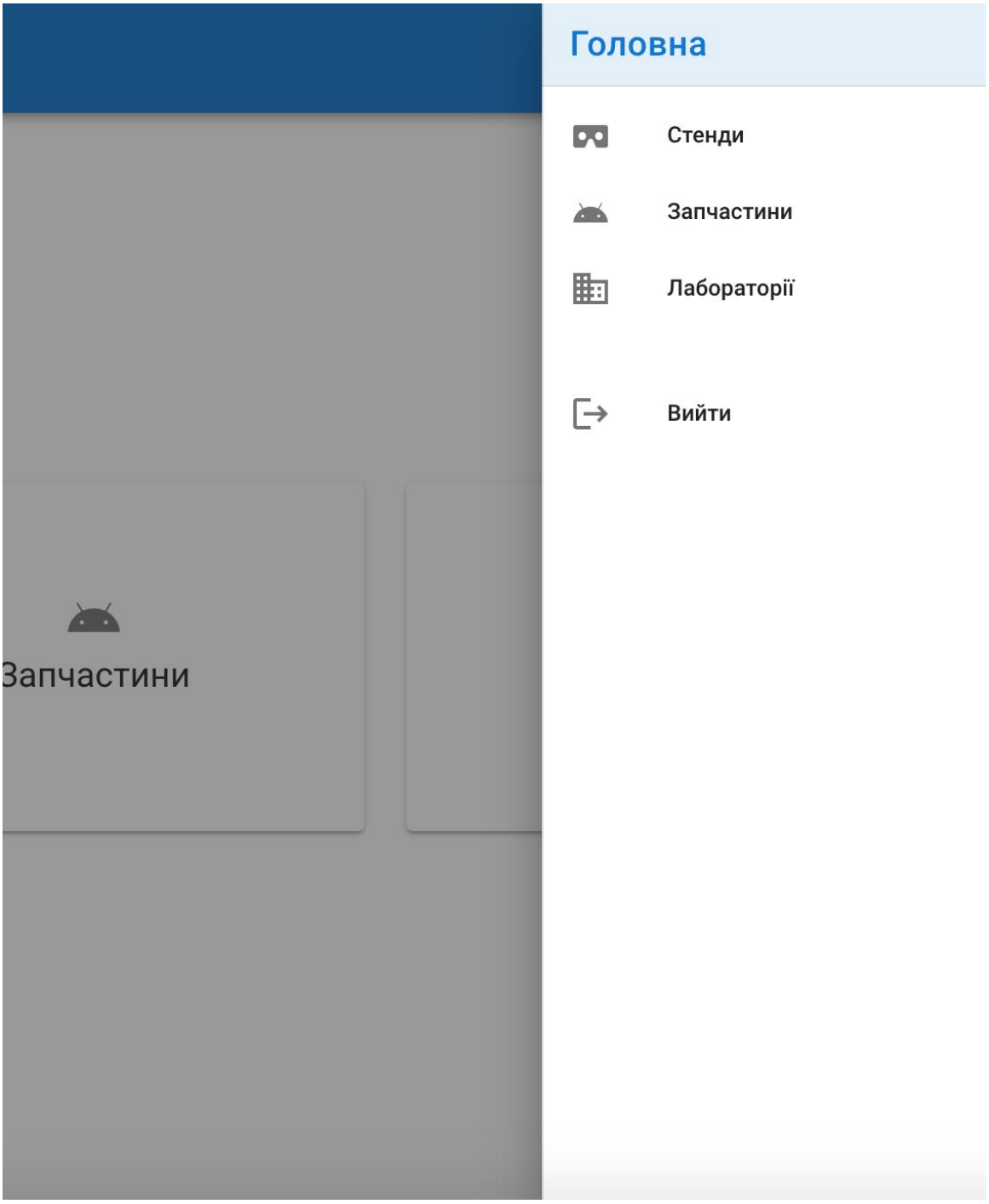

Рисунок 8.5 — Випадаюче меню

При виборі категорії стенди, користувач зможе переглянути всі доступні стенди (рисунок 8.6), а також створити новий, при натисканні на іконку “новий

стенд” (рисунок 8.7). Також користувач може видалити непотрібний йому стенд натиснувши кнопку з хрестиком справа зверху кожного стенду.

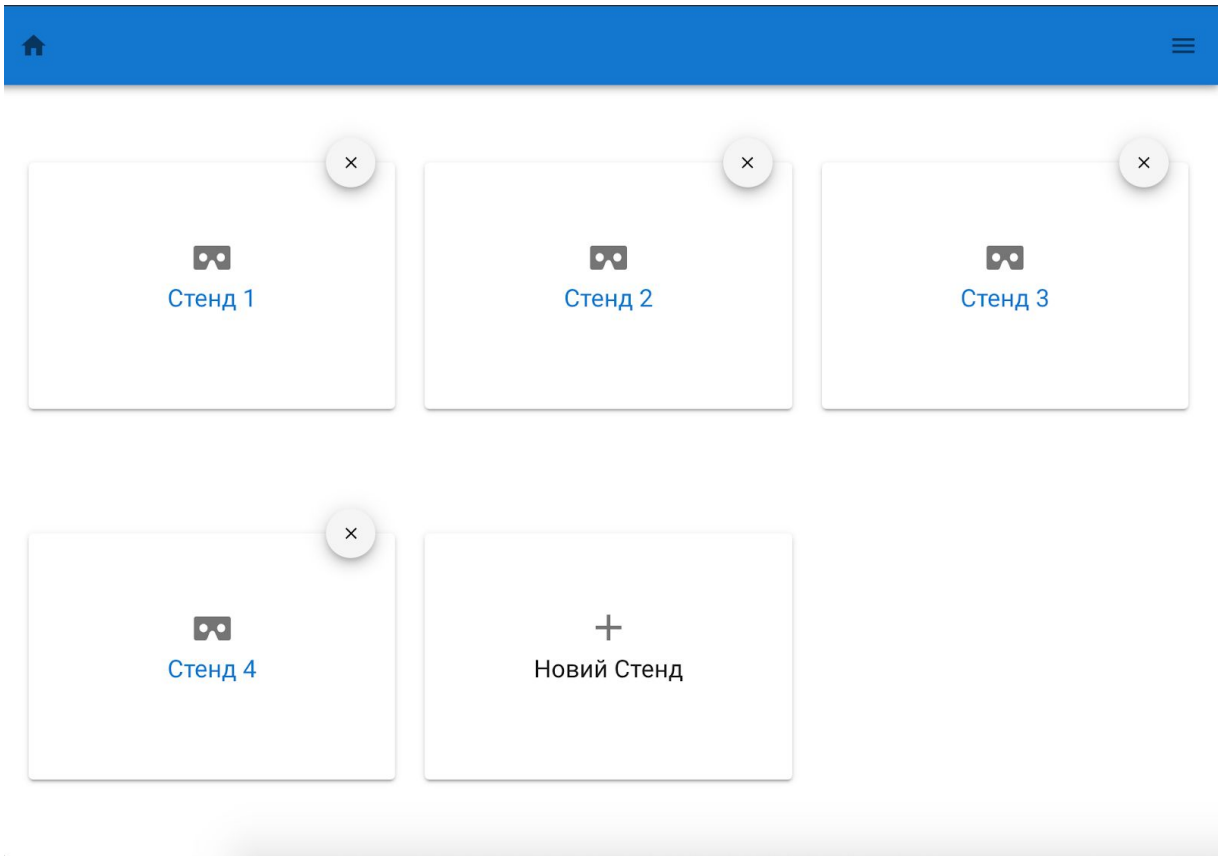

Рисунок 8.6 — Всі стенди

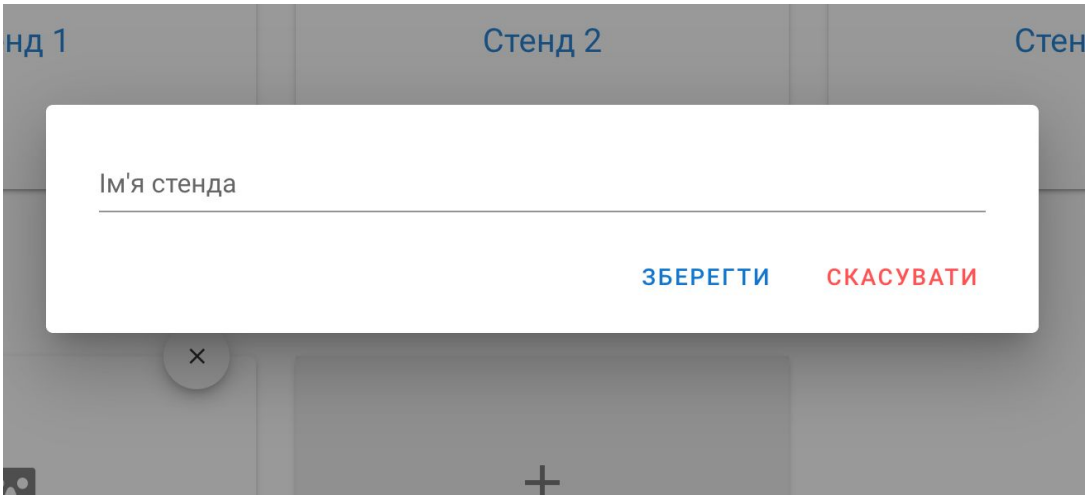

Рисунок 8.7 — Створити новий стенд

Далі перейдемо в розділ одного стенда (рисунок 8.8).

В ньому можна переглянути всі деталі цього стенда, додавати нові, редагувати їх та видаляти.

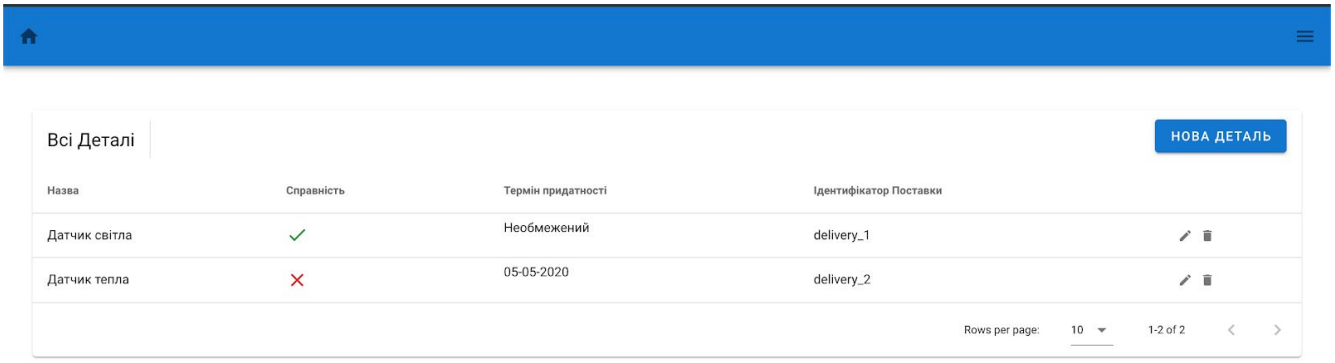

Рисунок 8.8 — Управління деталями одного стенда

На цій же сторінці є функціонал перегляду статистики стенду. Для його використання необхідне наявне API, з визначеною структурою. Посилання на API вставляється у відповідне поле, для того щоб візуалізувати дані, які передаються (рисунок 8.9).

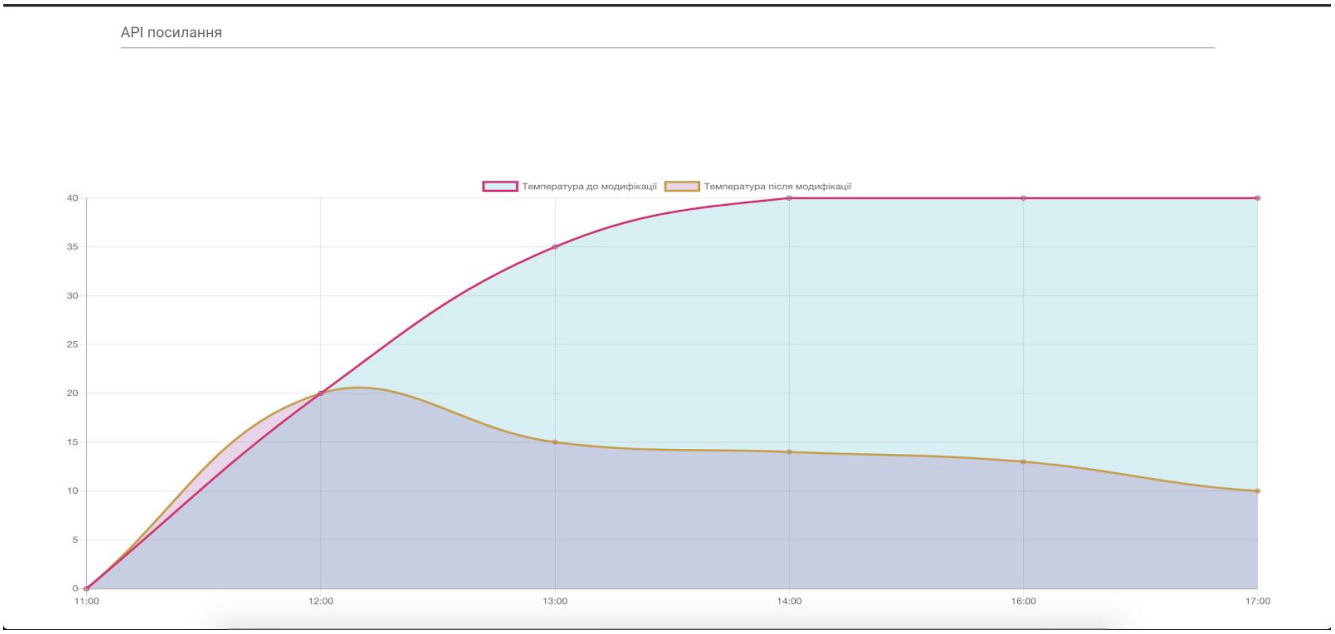

Рисунок 8.9 — відображення статистики

Для коректної роботи відображення, дані повинні приходити у форматі JSON, і виконувати наступні вимоги:

1. Після переведення JSON з рядкового типу він повинен мати тип об’єкт (а не тип масив).
2. Отриманий об’єкт повинен мати 2 поля — labels та datasets.
3. Поле label повинне мати масив назв пунктів для осі X.
4. Поле datasets повинне мати масив об’єктів графіків.

5. Кожен об’єкт графіка повинен мати поле label, яке має назву графіка, а також поле data, яке містить масив числових параметрів графіка.
6. Кількість елементів масиву для полів labels та datasets.data має бути однакова. Якщо немає даних для деяких пунктів осі X — рекомендується заповнити їх нульовими значеннями.

На практиці це дуже проста структура даних, вигляд якої можна переглянути на рисунку 8.10.

```
{
  labels: ["11:00", "12:00", "13:00", "14:00", "16:00", "17:00"],
  datasets: [
    {
      label: "Температура до модифікації",
      data: [0, 20, 35, 40, 40, 40, 40]
    },
    {
      label: "Температура після модифікації",
      data: [0, 20, 15, 14, 13, 10, 10]
    },
    {
      label: "Температура після встановлення системи охолодження",
      data: [0, 12, 15, 14, 13, 10, 10]
    }
  ]
}
```

Рисунок 8.10 — приклад структури даних

Наступний функціонал, який доступний в веб системі — це перегляд всіх деталей (рисунок 8.11).

На цій сторінці користувач може переглянути всі деталі які присутні в лабораторій, а також в яких стендах ці деталі задіяні.

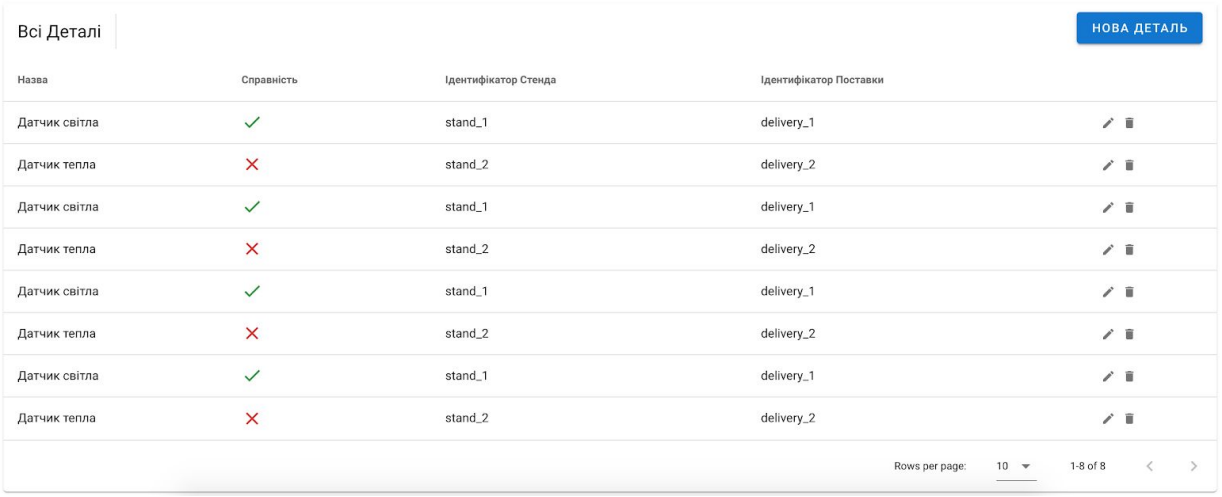

Рисунок 8.11 — Перегляд всіх деталей

До таблиці деталей можна добавляти нові (рисунок 8.12), редагувати та видаляти вже створені. При створенні або редагуванні деталі, її можна одразу закріпити за якимось стендом. Для цього створений випадаючий список усіх стендів (рисунок 8.13), з допомогою якого користувач може обрати необхідний йому стенд.

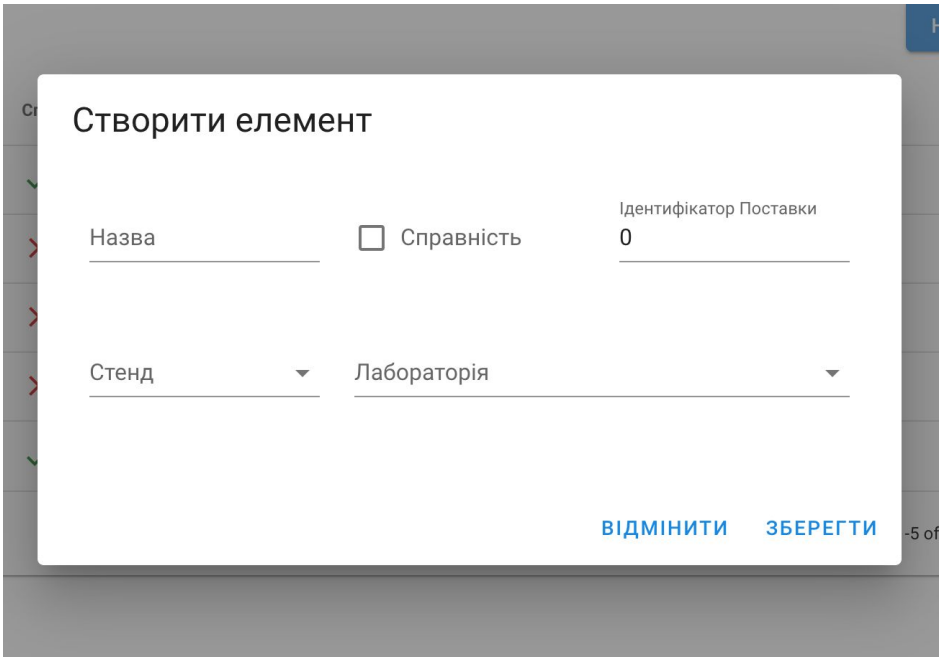

Рисунок 8.12 — Перегляд всіх деталей

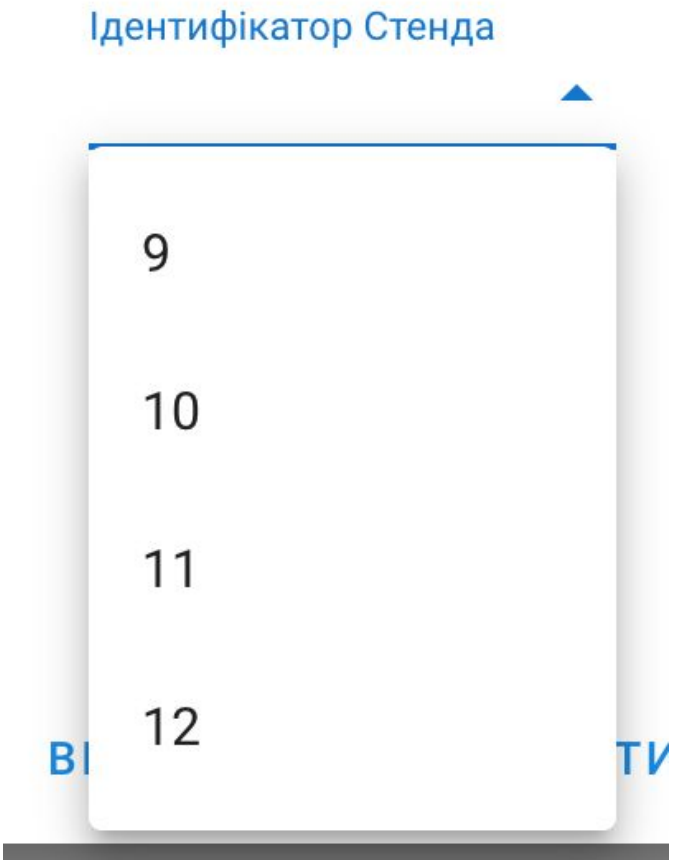

Рисунок 8.13 — Вибір стенда

Для того щоб видалити елемент таблиці треба буде підтвердити свою дію (рисунок 8.14). Аналогічне підтвердження необхідно виконати і для видалення інших елементів - лабораторій і стендів. Це звичайна практика, і зроблено для того, щоб уникнути видалень через випадкове натискання.

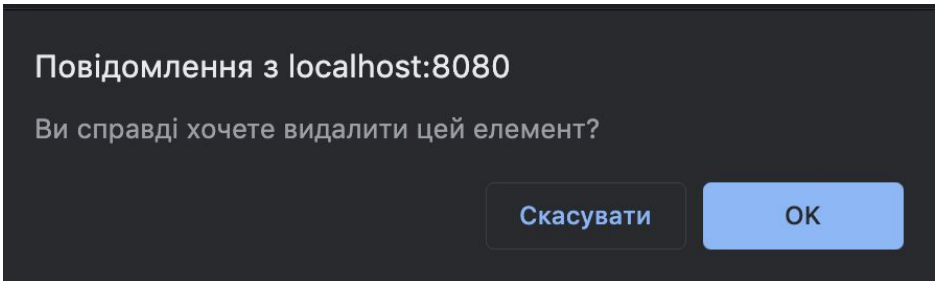

Рисунок 8.14 — Підтвердження видалення

Також варто зауважити, що в кожна таблиця має зручне управління. Можна сортувати дані по вибраному полю, а також керувати пагінацією таблиці.

Остання задача веб-системи — керування лабораторіями.

Інтерфейс всіх лабораторій виглядає аналогічно до інтерфейсу всіх стендів (рисунок 8.6). На цій сторінці можна переглянути всі лабораторії і добавити нові при потреби, знову ж таки аналогічно до інтерфейсу додавання нового стенду (рисунок 8.7).

Перейшовши на сторінку лабораторії, ми можемо переглянути всі стенди і деталі які розташовані в цій лабораторії. Всі стенди та деталі лабораторії відображені списком, для зручності користувач може перейти на сторінку потрібного стенда, щоб переглянути більше інформації про нього. Також користувач може змінювати розташування стендів та деталей в межах лабораторії (рисунок 8.15).

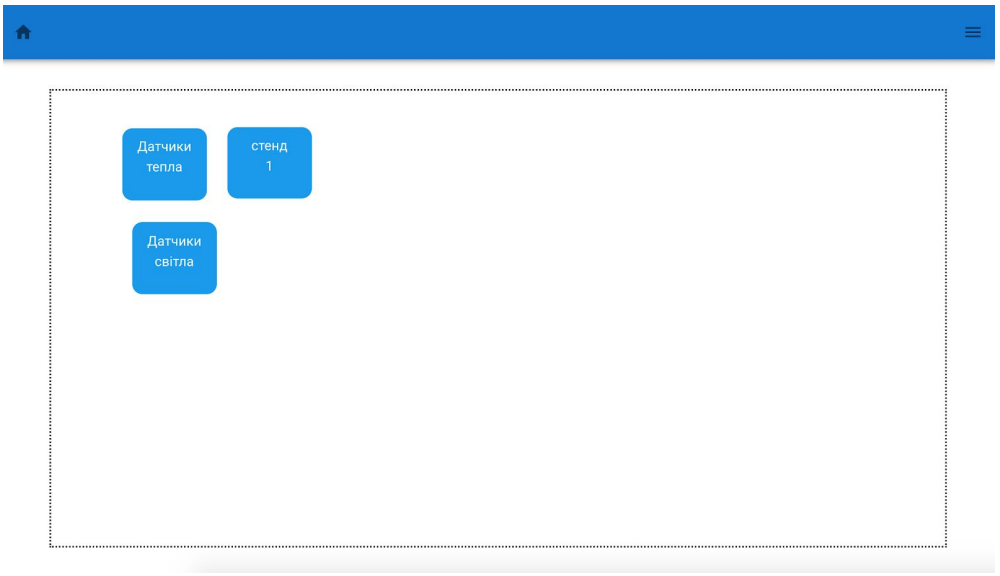

Рисунок 8.15 — переміщення стендів і деталей

Даний функціонал ще знаходиться в режимі бета версії, і потребує доопрацювання, проте ним вже можна користуватися.

З ростом користувачів смартфонів стрімко зростає і частка мобільного трафіку. Тому дуже важливо, щоб дизайн веб системи був адаптивним. Адаптивний дизайн (або адаптивна верстка)[7] – це особливий вид верстки сайту, який враховує характеристики різних пристроїв, забезпечуючи правильне відображення веб-ресурсу на екранах різного розміру. Таким чином, відвідувач може без проблем скористатися всіма можливостями сайту за допомогою свого

смартфону або планшету. Розглянемо кілька прикладів адаптивної верстки. Приклад того як виглядатиме таблиця на всіх деталей можна побачити на рисунку 8.16.

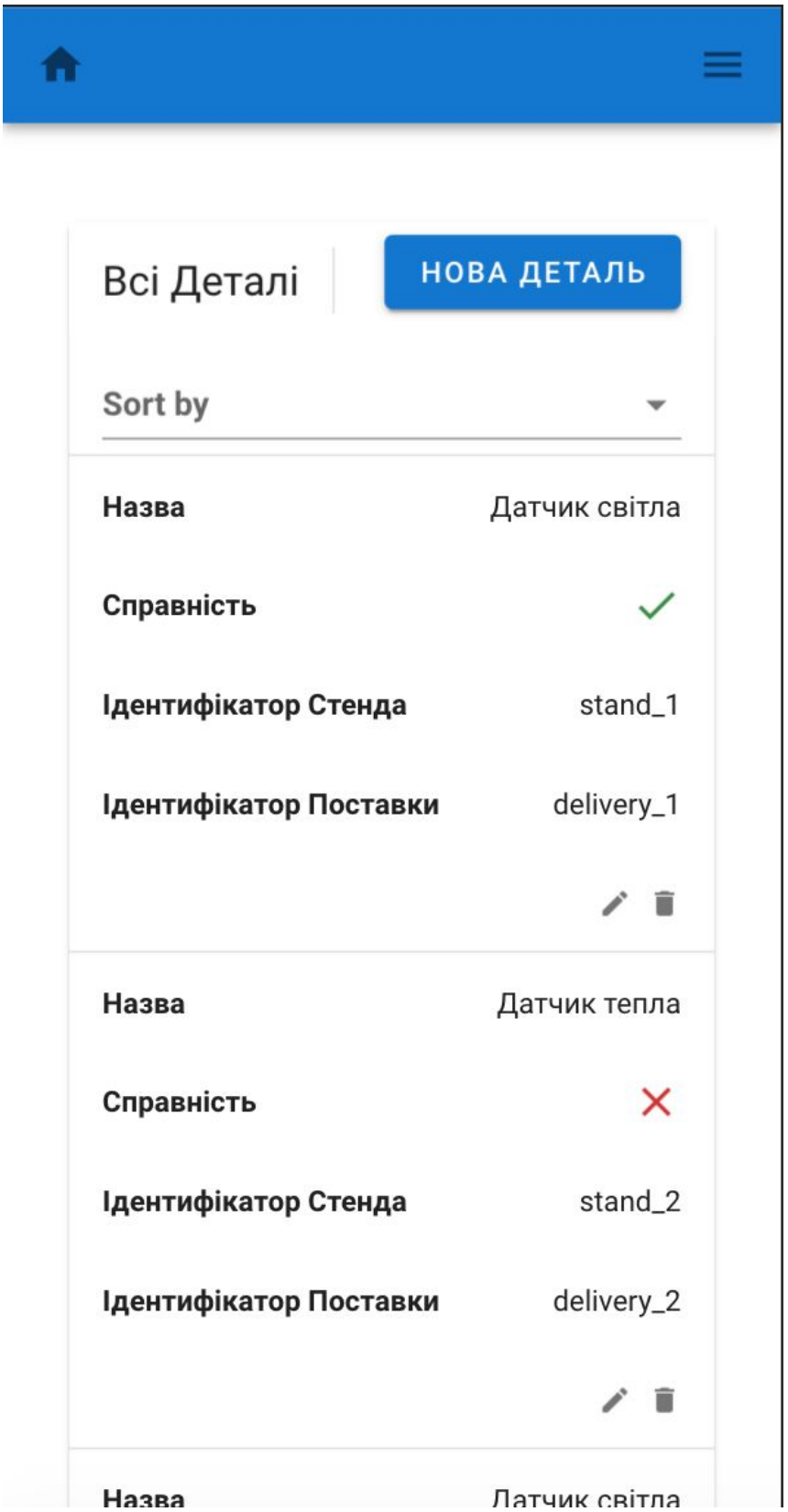

Рисунок 8.16 — вигляд таблиці на мобільному пристрої

У якості ще одного прикладу можна представити домашню сторінку користувача (рисунок 8.17), так як її структура є подібною до структури сторінок усіх стендів та лабораторій.

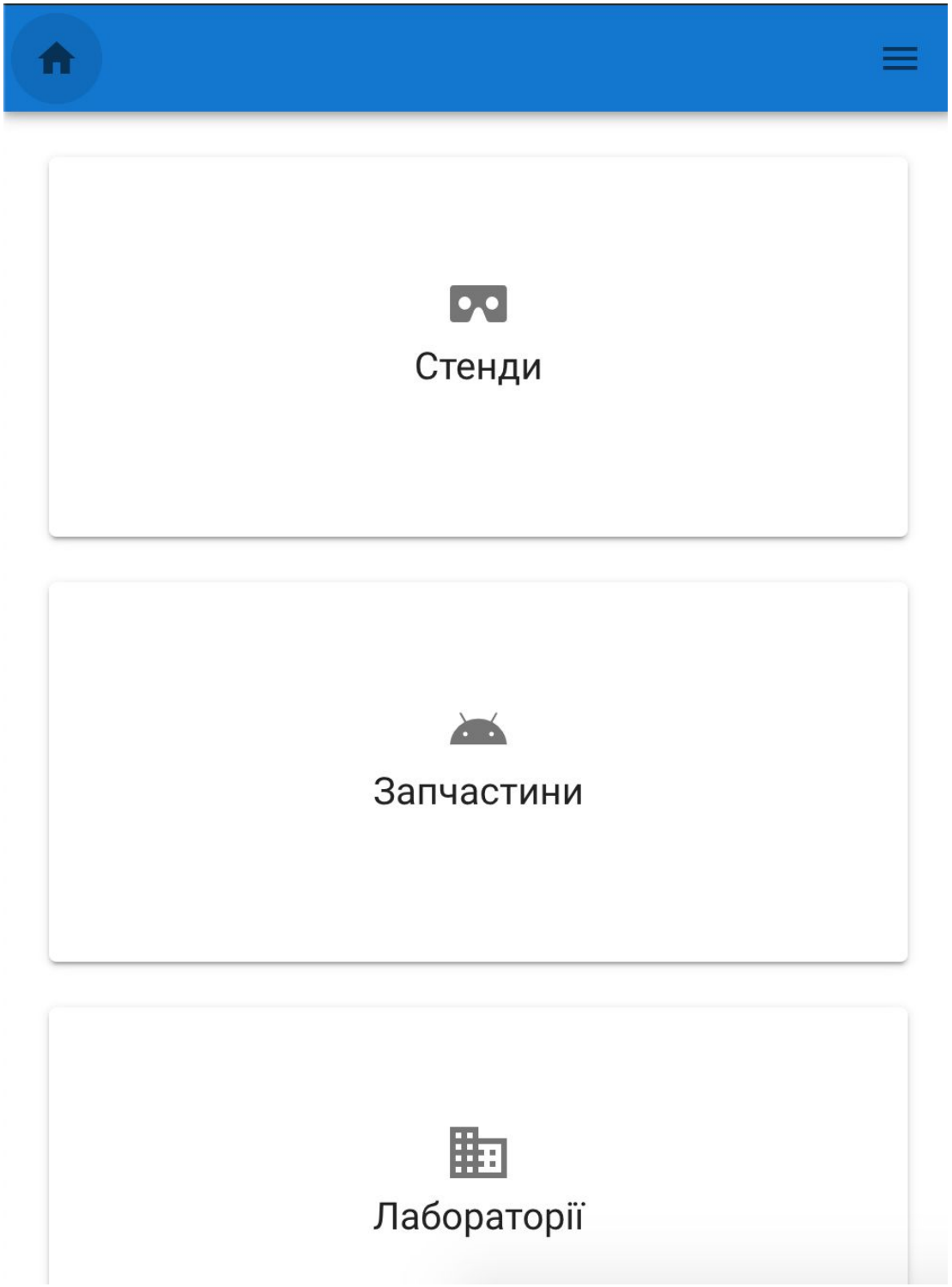

Рисунок 8.17 – інтерфейс домашньої сторінки на мобільному розширенні.

Такий інтерфейс є гнучким, та зручним для користувача з мобільним пристроєм.

9. Висновки

1. Був здійснений пошук та аналіз систем моніторингу експлуатації об’єктів. Вдалося визначити основні переваги та недоліки таких систем. Було виявлено, що на даний момент не існує системи моніторингу яка б відповідала вимогам відповідно до поставленої задачі.
2. В результаті дипломної роботи була розроблена інформаційно-довідкова система супроводження експлуатації лабораторних стендів.
3. Розроблена система є веб-застосунком, тому є кросплатформною.
4. Розроблена система надає функціонал моніторинг експлуатації стендів.
5. Розроблена система надає функціонал обліку запчастин.
6. Розроблена система надає функціонал встановлення розташування самих стендів і деталей в межах лабораторії.
7. Було використано різні підходи до проектування, зокрема використання патернів MVC та MVVM.

10. Список використаних джерел

1. Система моніторингу роботи “Dalgakiran” Режим доступу: http://dalgakiran.su/service/monitoring-raboti-sistemi
2. Офіційна документація фреймворка Vue.js — старт розробки. Режим доступу: https://012.vuejs.org/guide/
3. Офіційна документація сховища даних Vuex — старт розробки. Режим доступу: https://vuex.vuejs.org/ru/guide/
4. Офіційна документація фреймворка Laravel. Режим доступу: https://laravel.com/docs/7.x
5. Офіційна документація плагіна Vuetify. Режим доступу: https://vuetifyjs.com/en/
6. Офіційна документація фреймворка Vue.js — однофайловий компонент. Режим доступу: https://ru.vuejs.org/v2/guide/single-file-components.html
7. Адаптивний дизайн сайту. Режим доступу: https://ag.marketing/adaptyvnyy-dyzayn-saytu/
8. Об’єкт FormData. Режим доступу: https://developer.mozilla.org/ru/docs/Web/API/FormData
9. The Unified Modeling Language. UML-діаграми. Режим доступу: https://www.uml-diagrams.org/
10. Операції CRUD. Режим доступу: https://devman.org/encyclopedia/django_orm/CRUD/
11. JavaScript. Режим доступу: https://developer.mozilla.org/uk/docs/Web/JavaScript
12. PHP – загальна характеристика. Режим доступу: https://www.php.net/manual/ru/intro-whatis.php

13. Діаграма класів. Режим доступу: http://www.dut.edu.ua/ua/news-1-0-8002-zastosuvannya-uml-chastina-3-diagrama-klasiv----class-diagram
14. Офіційна документація по діаграмах класів. Режим доступу: https://www.uml-diagrams.org/class-diagrams-overview.html
15. Сервіс перевірки підтримки браузерів. Режим доступу: https://caniuse.com/
16. Рушії Javascript. Режим доступу: https://learn.javascript.ru/intro

Додаток 1

Інформаційно-довідкова система супроводження процесу експлуатації лабораторних стендів

Специфікація

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТІ6166_20Б

Аркушів 1

Позначення	Найменування	Примітки
Документація		
1	Пояснювальна записка.docx	н/п
2	Додаток 2. Текст програмного модуля	н/п
3	Додаток 3. Опис програмного модуля	н/п
Компоненти		
1	Роутинг REST API	Визначає типи запитів, передані параметри, а також вказує необхідність авторизації через middleware
2	Моделі Laravel	Зв’язуються з базою даних, здійснюють всі необхідні операції бізнес логіки
3	Контроллери Laravel	Здійснюють зв’язок в виглядом, викликають всі необхідні функції моделі
4	Моделі Vue	Зберігають стан системи, виконують запити на сервер, керують станом
5	Модель вигляду	Прослуховує події викликані користувачем, відслідковує реактивні дані
6	Вигляд Vue	Відображає інтерфейс користувача

Додаток 2

Інформаційно-довідкова система супроводження процесу експлуатації лабораторних стендів

Текст програми

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТІ6166_20Б

Аркушів 7

Київ - 2020

2.1 Vuex

```
import Vue from 'vue'
import Vuex from 'vuex'
import http from '../plugins/http'
import formDataBuilder from "../plugins/formDataBuilder";

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    access_token: "",
    user: {},
    stands: [],
    laboratories: [],
    details: []
  },
  getters: {
    getStandsId(state) {
      return state.stands.map(item => item.id);
    },
    getLaboratoriesId(state) {
      return state.laboratories.map(item => item.id);
    }
  },
  mutations: {
    setToken(state, access_token) {
      state.access_token = access_token;
    },
    mutate(state, payload) {
      state[payload.key] = payload.value;
    },
    setLaboratories(state, laboratories) {
      state.laboratories = laboratories;
    },
    setStands(state, stands) {
      state.stands = stands;
    },
    setDetails(state, details) {
      console.log("asda");

      state.details = details;
    }
  },
  actions: {
    async loginUser({ commit }, user) {
      await http.post('/user/login', formDataBuilder(user))
        .then((response) => {
```

```
      if (response.data.access_token) {
        localStorage.setItem("access_token", response.data.access_token)
        commit('setToken', response.data.access_token)
        http.defaults.headers.common['Authorization'] = 'Bearer ' + response.data.access_token;
        console.log(http.defaults.headers.common['Authorization']);
      }
    })
},
async getUser({ commit }) {
  await http.get('/user')
    .then((response) => {
      console.log(response.data);
    })
  await http.get('/laboratories')
    .then((response) => {
      commit("setLaboratories", response.data);
    })
  await http.get('/stands')
    .then((response) => {
      commit("setStands", response.data)
    })
  await http.get('/details')
    .then((response) => {
      commit("setDetails", response.data)
    })
},
async createNewUser({ }, user) {
  await http.post('/user/create_user', formDataBuilder(user))
    .then((response) => {
      console.log(response.data);
    })
},
async createLaboratory({ commit }, name) {
  await http.post('/create/laboratory', formDataBuilder({ name }))
    .then((response) => {
      commit("setLaboratories", response.data);
    })
},
async deleteLaboratory({ commit }, id) {
  await http.delete('/delete/laboratory', { params: { id } })
    .then((response) => {
      commit("setLaboratories", response.data);
    })
},
async createStand({ commit }, name) {
  await http.post('/create/stand', formDataBuilder({ name }))
    .then((response) => {
      commit("setStands", response.data);
```

```
      })
    },
    async deleteStand({ commit }, id) {
      await http.delete('/delete/stand', { params: { id } })
        .then((response) => {
          commit("setStands", response.data)
        })
    },
    async createDetail({ commit }, detail) {
      console.log(detail);
      console.log(formDataBuilder(detail));
      await http.post('/create/detail', formDataBuilder(detail))
        .then((response) => {
          commit("setDetails", response.data);
        })
    },
    async deleteDetail({ commit }, id) {
      await http.delete('/delete/detail', { params: { id } })
        .then((response) => {
          commit("setDetails", response.data);
        })
    },
  }
})
```

2.2. formDataBuilder.js

```
function formDataBuilder(dataObject) {
    let formData = new FormData();
    for (const key in dataObject) {
        formData.set(key, dataObject[key]);
    }
    return formData;
}

export default formDataBuilder;
```

2.3. http.js

```
import axios from 'axios'

const http = axios.create({
    baseURL: 'http://127.0.0.1:8000/api',
    'Content-Type': 'multipart/form-data'
})
```

```
export default http;
```

2.4. http.js

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import RegistrationPage from '../views/RegistrationPage.vue'
import HomePage from '../views/HomePage.vue'
import StandsPage from '../views/StandsPage.vue'
import DetailsPage from '../views/DetailsPage.vue'
import LaboratoriesPage from '../views/LaboratoriesPage.vue'
import OneStand from '../views/OneStand.vue'
import OneLaboratory from '../views/OneLaboratory.vue'


Vue.use(VueRouter)

const routes = [
 {
   path: '/',
   name: 'registration',
   component: RegistrationPage
 },
 {
   path: '/home',
   name: 'home',
   component: HomePage
 },
 {
   path: '/stands',
   name: 'stands',
   component: StandsPage
 },
 {
   path: '/details',
   name: 'details',
   component: DetailsPage
 },
 {
   path: '/laboratories',
   name: 'laboratories',
   component: LaboratoriesPage
 },
 {
   path: '/stands/:id',
```

```
    name: 'one_stand',
    component: OneStand
  },
  {
    path: '/laboratory/:id',
    name: 'one_laboratory',
    component: OneLaboratory
  }
]

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})


export default router
```

2.5. RegistrationPage.vue

```
<template>
  <v-app id="inspire">
    <v-content>
      <v-container class="fill-height" fluid>
        <v-row align="center" justify="center">
          <v-col cols="12" sm="8" md="4">
            <v-card class="elevation-12">
              <v-toolbar color="primary" dark flat>
                <v-toolbar-title v-if="is_registration">Зареєструйтесь</v-toolbar-title>
                <v-toolbar-title v-else>Увійдіть в систему</v-toolbar-title>
                <v-spacer />
              </v-toolbar>
              <v-card-text>
                <v-form>
                  <v-text-field
                    label="Ім'я"
                    name="login"
                    prepend-icon="person"
                    type="text"
                    v-if="is_registration"
                    v-model="name"
                    :rules="nameRules"
                  />
                  <v-text-field
                    label="Email"
```

```
              name="login"
              prepend-icon="person"
              type="email"
              v-model="email"
              :rules="emailRules"
            />
            <v-text-field
              id="password"
              label="Пароль"
              name="password"
              prepend-icon="lock"
              type="password"
              v-model="password"
              :rules="passwordRules"
            />
          </v-form>
        </v-card-text>
        <v-card-actions>
          <v-spacer />
          <v-btn
            color="primary"
            @click="loginUser"
          >{{!is_registration ? "Увійти" : "Зареєструватись"}}</v-btn>
        </v-card-actions>
      </v-card>
    </v-col>
  </v-row>
  <v-col justify="center" align="center">
    <p v-if="is_registration">Увійдіть, якщо ви вже зареєстровані</p>
    <p v-else>Або зареєструйтесь, якщо ви ще цього не зробили</p>
    <v-btn
      @click="toggleRegistration"
      color="primary"
    >{{is_registration ? "Увійти" : "Зареєструватись"}}</v-btn>
  </v-col>
 </v-container>
 </v-content>
 </v-app>
</template>

<script>
export default {
 name: "RegistrationPage",
 data: () => ({
  is_registration: false,
  email: "",
  password: "",
  name: "",
```

```
    nameRules: [v => !!v || "Ім'я є обов'язковим"],
    emailRules: [v => !!v || "Логін є обов'язковим"],
    passwordRules: [v => !!v || "Пароль є обов'язковим"]
  }),
  methods: {
    toggleRegistration() {
      this.is_registration = !this.is_registration;
    },
    async loginUser() {
      if (this.is_registration) {
        await this.$store.dispatch("createNewUser", {
          email: this.email,
          password: this.password,
          name: this.name
        });
      } else {
        await this.$store.dispatch("loginUser", {
          email: this.email,
          password: this.password
        });
        await this.$store.dispatch("getUser");
        if (this.$store.state.access_token) {
          this.$router.push("/home");
        }
      }
    }
  },
  created() {
    let access_token = localStorage.getItem("access_token");
    if (access_token) {
      this.$store.commit("setToken", access_token);
      this.$router.replace("/home");
    }
  }
};
</script>
```

2.6. api.php

```
<?php

use Illuminate\Http\Request;

Route::middleware('auth:api')->get('/stands', 'StandsController@getStands');
Route::middleware('auth:api')->get('/laboratories', 'LaboratoryController@getLaboratories');
Route::middleware('auth:api')->get('/details', 'DetailController@getDetails');
```

```
Route::prefix('/create')->group( function () {
    Route::middleware('auth:api')->post('/stand', 'StandsController@createStand');
    Route::middleware('auth:api')->post('/laboratory', 'LaboratoryController@createLaboratory');
    Route::middleware('auth:api')->post('/detail', 'DetailController@createDetail');

});
Route::prefix('/delete')->group( function () {
    Route::middleware('auth:api')->delete('/stand', 'StandsController@deleteStand');
    Route::middleware('auth:api')->delete('/laboratory', 'LaboratoryController@deleteLaboratory');
    Route::middleware('auth:api')->delete('/detail', 'DetailController@deleteDetail');
});
Route::prefix('/update')->group( function () {
    Route::middleware('auth:api')->delete('/stand', 'StandsController@updateStand');
    Route::middleware('auth:api')->delete('/laboratory', 'LaboratoryController@updateLaboratory');
    Route::middleware('auth:api')->delete('/detail', 'DetailController@updateDetail');
});


Route::prefix('/user')->group( function () {
    Route::middleware('auth:api')->get('/', 'UserController@getUser');
    Route::post('/login', 'api\v1\LoginController@login');
    Route::post('/create_user', 'UserController@createUser');
});
```

Додаток 3

Інформаційно-довідкова система супроводження процесу експлуатації лабораторних стендів

Опис програми

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТІ6166_20Б

Аркушів 1

Київ - 2020

АНОТАЦІЯ

Розроблений програмний продукт є інформаційно-довідковою системою супроводження процесу експлуатації лабораторних стендів. Така система є веб застосунком, а тому є кросплатформенною. Вона створена у вигляді SPA,а тому є швидкою і зручною у використанні. Також серверна частина застосунку представлена у вигляді REST API, що дає простір для розробки у майбутньому також і мобільного додатку. Для розробки клієнтської частини застосунку був використаний JavaScript фреймворк Vue.js, з використанням плагіну Vuetify. Для розробки серверної частини був обраний PHP фреймворк Laravel, і MySQL в якості бази даних.

У функціонал системи входить облік деталей, моніторинг лабораторних стендів, а також відслідковування місця розташування стендів та деталей в межах лабораторії. Також лабораторним стендам можна додавати API, яке буде формувати графіки на основі отриманих від нього статистичних даних.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ ТА ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Інформаційно-довідкова система супроводження процесу експлуатації лабораторних стендів була розроблена з використанням мов програмування JavaScript та PHP. Для клієнтської частини був використаний фреймворк Vue.js, з використанням різноманітних плагінів:

- Vuetify – плагін для оформлення інтерфейсу користувача, який містить багато стандартних елементів інтерфейсу, які легко стилізуються директивами без використання в явному вигляді CSS стилів.
- Axios – бібліотека для створення AJAX запитів. Полегшує роботу з запитами на сервер, дає зручні методи для встановлення хедерів, get-параметрів і інших загальних налаштувань.
- Chart.js – бібліотека для створення інтерактивних графіків. В даному випадку використовувалась надбудова vue-chartjs, яка створена для роботи з фреймворком Vue.js, та підтримує реактивність даних.
- Interact.js – бібліотека для drag-and-drop технології. Дає зручний вбудовану функцію для переміщення DOM елементів в межах батьківського елемента.

Для серверної частини використовується лише два плагіни:

- laravel-cors – бібліотека для налаштування міждоменних запитів.
- passport – бібліотека для реалізації авторизації з використанням токенів. Дає зручні методи для роботи з авторизованим користувачем, а також створює всі необхідні таблиці в базі даних.

Функціонал системи можна поділити на три основні складові: облік деталей, моніторинг лабораторних стендів, а також відслідковування місця розташування стендів та деталей в межах лабораторії.

Користувач може створювати, редагувати, видаляти деталі. При перегляді таблиць деталей, їх можна сортувати, та встановлювати пагінацію в межах таблиці.

Для лабораторних стендів можна додавати API, яке буде формувати графіки на основі отриманих від нього статистичних даних. Також зі сторінки стендів можна керувати деталями, які наявні в цьому стенді.

І останній функціонал – це керування лабораторіями. Кожен користувач може створювати лабораторії і розподіляти стенди і деталі в їх межах.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Програмний продукт складається з клієнтської частини з використанням паттерну MVVM (Model-View-ViewModel), серверної частини з використанням патерну MVC (Model-View-Controller). Сполучення серверної і клієнтської частини відбувається через REST API (рисунок 1).

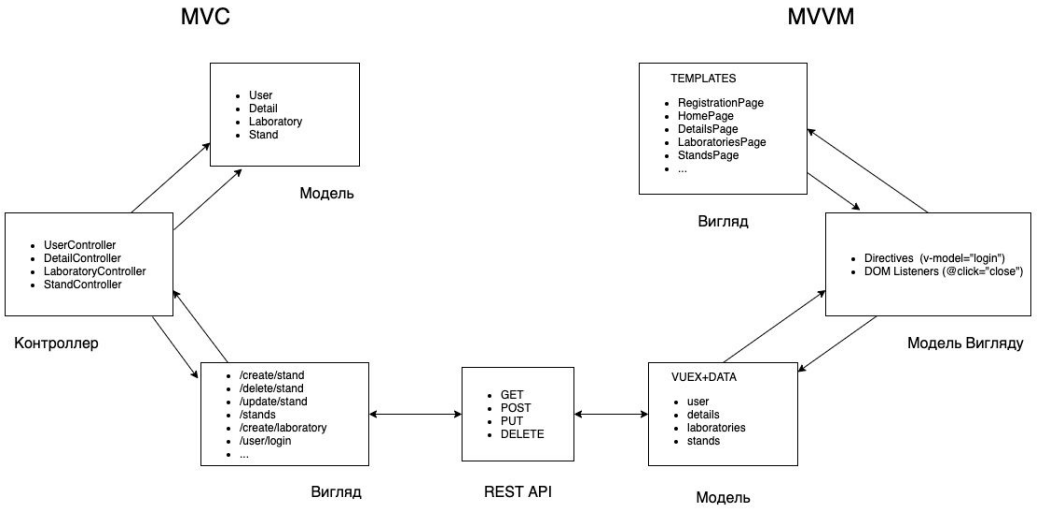

Рисунок 1 – Архітектура системи.

У клієнтської частини модель представлена сховищем даних Vuex та локальним станом data, вигляд представлений шаблонами, а модель вигляду представлена директивами та обробниками подій. Дані на клієнтській частині (в даному випадку це лише токен), ми зберігаємо в сховищі браузера LocalStorage.

У серверної частини модель представлена класами, які відображають сутності баз даних, для кожного класу моделі створений свій контроллер, який викликає методи моделі, а виглядом є шляхи зв’язку з клієнтською частиною. Дані на серверній частині зберігаються в базі даних MySQL.

І як зображено на діаграмі REST API, який включає 4 методи: get, post, put, delete, і зв’язує клієнтську частину з серверною.

Останнім пунктом системи є MySQL базою даних. База даних представлена основними чотирма таблицями – таблиці користувачів, стендів, деталей та лабораторій.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Для розробки системи використовувався текстовий редактор Visual Studio Code. Для роботи з запитами використовувалась програма Postman. Для роботи з Vue.js фреймворком використовувалась платформа Node.js, а для розгортання проекту на Laravel – веб-сервер Apache. Для роботи з базою даних використовувався інструмент phpMyAdmin.

Сама система не потребує інсталяції, а запускається через браузер. Система може використовуватись на різних операційних системах. Для запуску підходять різні види браузерів, наприклад Google Chrome, Mozilla Firefox, Safari, Opera і тд.

ВХІДНІ І ВИХІДНІ ДАНІ

Майже всі вхідні дані приходять в систему від користувача. Користувач в ході використання системи заповнює інформацію про стенди, деталі і лабораторії з клавіатури. Деякі поля пропонують вибір зі списку, або чекбокс. Дані, які вводяться користувачем з клавіатури валідуються на стороні сервера та на стороні клієнта. Також вхідними даними є позиція деталей та стендів в межах лабораторії. Вона задається через перетягування елементів користувачем.

Другим типом вхідних даних, є дані з введеного API для стендів. Стенд може вводити посилання на своє власне API, яке повинне мати визначену структуру. При вірних параметрах система приєме дані зі стороннього API, і на їх основі промалює графіки.

Вихідними даними є списки стендів та лабораторій, таблиці деталей, розташування елементів в межах лабораторій, і графіки стендів.